Monad as modality

Satoshi Kobayashi

Department of Applied Mathematics and Informatics, Ryukoku University, Seta, Ohtsu, 520-21, Japan

Abstract

In 1989, Eugenio Moggi proposed a categorical framework for program semantics based on the notion of a strong monad. He showed that various kinds of computation can be modeled in his framework. On the other hand, strong monads are not suited for the categorical semantics of traditional modal logics. According to these observations, Moggi thought that the Curry-Howard correspondence would not hold between programs and constructive proofs in modal logics. However, contrary to his view, we can show that proofs in a certain kind of modal logics are actually considered as programs. In this paper, first we shall introduce the notion of an \( \mathcal{L} \)-strong monad which is a generalization of strong monads. Using this new notion, we can generalize Moggi's semantics-preserving soundness and completeness with respect to his equational logic. Next we shall show that \( \mathcal{L} \)-strong monads give a sound and complete semantics of a constructive version of S4 modal logic. Finally, we present a method to extract a monad-based imperative functional program from a proof in the modal logic. Interestingly, this method can also be understood in terms of \( \mathcal{L} \)-strong monads.

1. Introduction

In 1989, Eugenio Moggi proposed a categorical framework for program semantics based on the notion of a strong monad. His framework is so general that it can treat various kinds of computation such as non-terminating evaluation, non-deterministic computation, side-effects, exceptions, continuation passing, I/O, etc. Today, monad-based imperative functional programming is extensively studied.

It is known that monads can be used to interpret logical modalities such as possibility in modal logic or ‘why not’ of linear logic. Comonads, the dual of monads, are used to model necessity or ‘of course’ modality. However, strong monads are not suitable for modeling modalities. The reason is that if we interpret modalities in a strong monad, then some unnatural formulas such as \( A \land \diamond B \to \diamond (A \land B) \) become valid. In this sense, strong monads are too strong. However, simple monads are too weak as the basis of program semantics. According to these observations, Moggi thought that the Curry–
Howard correspondence would not hold between programs and constructive proofs in modal logics. If it is correct, the formal program derivation techniques based on the proofs-as-programs principle will not be generalized to derive monad-based imperative programs.

However, fortunately, the fact is contrary to his conjecture. We can show that proofs in a certain kind of modal logics are actually considered as programs.

In this paper, we introduce the notion of an Z-strong monad as a generalization of a strong monad. This new notion gives a semantical framework suitable for both programs and modal logics. We can generalize Moggi's semantics-preserving soundness and completeness with respect to his equational logic. Moreover, Z-strong monads give a sound and complete semantics of a constructive version of S4 modal logic called CS4. We shall also present a type theory TCS4 which is a term assignment system for CS4. TCS4 is considered as an extension of the type system of Moggi's computational metalanguage. We define a collapsing map from TCS4 to an extended version of Moggi's type system. It will offer a clear understanding of the relation between these two type systems.

To establish the proofs-as-programs principle, we shall define a realizability interpretation of our modal logic. A realizer of a program specification formula is considered as a program which meet the specification. We can find such a realizer by collapsing a proof term of TCS4 using our collapsing map. This result gives an effective method to extract a monad-based imperative functional program written in the computational metalanguage from a proof in our modal logic. Interestingly, this method can also be understood in terms of Z-strong monads.

This paper is organized as follows: In Section 2, we shall review Moggi's program semantics based on strong monads. In Section 3, we consider an extension of the computational metalanguage, which will be used later to define a realizability interpretation of our modal logic. Then we shall discuss the incompatibility of strong monads with traditional modal logics in Section 4. In Section 5, we introduce the notion of an Z-strong monad, and using it we generalize Moggi's semantics. We prove soundness and completeness for our new semantics. In Section 6, we shall present a constructive version of S4 modal logic called CS4 and its type theoretical counterpart TCS4. We show that Z-strong monad gives a complete semantics of CS4 and TCS4. We define a collapsing map from TCS4 to the extended metalanguage. In Section 7, we define a realizability interpretation of CS4 and prove its soundness. Using this result, we show that proofs in CS4 (or TCS4) can be considered as programs. Further, we prove that the realizability interpretation is a special case of the categorical interpretation of our modal logic. Finally, we shall give a toy example of program extraction in Section 8.

In this manuscript, we assume that the reader has basic knowledge of: cartesian closed categories (ccc's), (strong) monads, comonads, intuitionistic and modal logics, realizability interpretations and constructive type theories. The reader unfamiliar with ccc's, monads or comonads is referred to MacLane's textbook [5]. For strong monads, see [6] or [7]. For intuitionistic logics, realizability interpretations and constructive type
theories, see [10]; a simple example of collapsing map is also found in Vol. II of this book. Goldblatt's book [3] will be convenient for computer scientists as an introduction to modal logics.

2. Review of Moggi's semantics

The central idea of Moggi's program semantics is that we distinguish the object $A$ of values of type $A$ from the object $MA$ of computations of type $A$, where $A$ and $B$ are the semantical domains corresponding to $A$ and $B$ respectively. An "impure" function $f$ from $A$ to $B$ is modeled by a morphism from $A$ to $MB$. Intuitively, an element $\alpha$ of $MA$ is a computation which may produce a value in $A$ as the resulting value; $\alpha$ itself is not a value in $A$. $\alpha$ and its resulting value are not identified. $M$ is called a notion of computation.

An important discovery of Moggi is that, in many interesting cases, $M$ and the associated operations form a strong monad structure. Moggi found a general framework of program semantics based on the notion of a strong monad.

In what follows, we write $f;g$ for the composition of two arrows $f:A \rightarrow B$ and $g:B \rightarrow C$, i.e. $f;g$ means $g \circ f$. It is well known that monads and Kleisli triples are in one-to-one correspondence. When a monad $(M,\eta,\mu)$ is given, we write $f^*$ for the Kleisli lifting of $f$ in the sense of the corresponding Kleisli category.

Side-effect monads: A side-effect monad is a typical example of strong monad. Here we discuss non-deterministic ones and deterministic ones. They will be important in later sections.

A non-deterministic side effect monad $(M,\eta,\mu,t)$ over Set (the category of sets) is defined as follows: Let $S$ be the set of states; then define $M(-) = \mathcal{P}(S \times (- \times S))$,

$$(s,(a',s')) \in \eta_A(a) \Leftrightarrow a' = a \land s' = s,$$

$$(s,(c'',s'')) \in \mu_A(c) \Leftrightarrow \exists c'.\exists s'.(s,(c',s')) \in c \land (s',(c'',s'')) \in c',$$

$$(s,(a',s')) \in t_{A,B}(a,b) \Leftrightarrow \exists b'.((s,(b',s')) \in b \land a' = (a,b')),$$

where $\mathcal{P}$ is the covariant power set functor. Then we have

$$\eta_A : A \rightarrow MA,$$

$$\mu_A : M^2A \rightarrow MA,$$

$$t_{A,B} : A \times MB \rightarrow M(A \times B).$$

The intuitive meaning of $(s,(a',s')) \in a$ is that the evaluation of $a$ that is started at the state $s$ can terminate at the state $s'$ and return $a'$ as the resulting value. The computation of $\eta_A(a)$ immediately returns $a$ and does not change the state. The computation of $\mu_A(a)$ is done as follows: first $a$ is computed; if it returns a value $a'$, then $a'$ is
computed; and if \( a' \) returns \( a'' \), then \( a'' \) is returned as the resulting value of \( \mu_A(a) \). To compute \( t_{A,B}(a,b) \), first compute \( b \) and receive a resulting value \( b' \), then return \( (a,b') \) as the resulting value of the whole computation.

The deterministic version of the side-effect monad can be defined on any ccc. For example, take \( \mathcal{C} = \text{CPO} \) (the category of complete partial orders). Let \( S \) be an object. Define

\[
M(-) = (S \Rightarrow (- \times S)),
\]

\[
\eta_A = \Lambda(id_{A\times S}) = \lambda s : S.(a,s),
\]

\[
\mu_A = \Lambda(\text{eval}_{S,(S\Rightarrow A\times S)}; \text{eval}_{S,A\times S}) = \lambda f : M^2 A.\lambda s : S.\text{eval}(fs)
\]

(where \( \text{eval}(f',s') = f's' \)),

\[
t_{A,B} = \Lambda(\alpha_{A,S\Rightarrow B\times S}; (id_A \times \text{eval}_{S,B\times S}); \alpha_{A,B,S}^{-1})
\]

\[
= \lambda (a,f) : A \times MB.\alpha : S.(\text{let } (b,s') = fs \text{ in } \langle(a,b),s'\rangle),
\]

where \( \Lambda \) is the currying operation and \( \text{eval} \) is the evaluation map; \( \lambda \)-means the informal lambda abstraction; \( \alpha \) is a natural isomorphism

\[
\alpha_{A,B,C} : (A \times B) \times C \rightarrow A \times (B \times C).
\]

For other examples of strong monads such as the monad of partiality or the monad of continuations, see [6] or [7]. Some practical applications of monads are found in [11, 12].

2.1. Computational metalanguage

In [7], Moggi proposed a formal language called computational metalanguage (the metalanguage, for short). The approach taken in [7] is as follows: first he establishes a clear categorical semantics of the metalanguage, and then describe the semantics of other languages in terms of the metalanguage.

**Types:** Assume that the set of basic types is given. Then the types of the language are defined by the following rules:

\[
\frac{}{\vdash A\ \text{type}} \quad (A \text{ is a basic type})
\quad \frac{}{\vdash \tau\ \text{type}} \quad \frac{}{\vdash M\tau\ \text{type}}
\quad \frac{\vdash \tau_1\ \text{type} \quad \vdash \tau_2\ \text{type}}{\vdash \tau_1 \times \tau_2\ \text{type}}
\quad \frac{\vdash \tau_i\ \text{type} \quad (1 \leq i \leq n)}{x_1 : \tau_1, \ldots, x_n : \tau_n \vdash x_i : \tau_i}
\quad \Gamma \vdash * : 1
\]

**Terms:** Suppose that the set of function symbols is given and each function symbol has its arity of the form \( \tau_1 \rightarrow \tau_2 \), where \( \tau_1 \) and \( \tau_2 \) are types. Then terms are defined by the following rules:
\[ \Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2 \quad \Gamma \vdash e : \tau_1 \times \tau_2 \]

\[ \Gamma \vdash \langle e_1, e_2 \rangle : \tau_1 \times \tau_2 \]

\[ \Gamma \vdash f(e_1) : \tau_2 \quad (f \text{ is a function symbol of arity } \tau_1 \to \tau_2) \]

\[ \Gamma \vdash e : \tau \quad \Gamma \vdash [e] : M\tau \]

\[ \Gamma \vdash e_1 : \tau M_1 \quad \Gamma, x : \tau_1 \vdash e_2 : \tau M_2 \]

\[ \Gamma \vdash (\text{let } x_1 \leftarrow e_1 \text{ in } e_2) : \tau M_2 \]

where \( \Gamma \equiv x_1 : \tau_1, \ldots, x_n : \tau_n \) is a typing context; \( x_1, \ldots, x_n \) are variables, and \( \tau_1, \ldots, \tau_n \) are types.

Intuitively, \( [a] : M\alpha \) is \( a : \alpha \) viewed as a computation, and the execution of \( (\text{let } x_1 \leftarrow e_1 \text{ in } e_2) \) is done as follows: first execute \( e_1 \) and bind \( x_1 \) to the resulting value \( v_1 \); then execute \( e_2 \) in this new binding environment and receive the resulting value \( v_2 \); finally, return \( v_2 \) as the resulting value of the whole execution.

### 2.2. Semantics

**Semantics of types:** The interpretation of types is given by \( \Theta = (\mathcal{C}, (M, \eta, \mu, \iota), \theta) \), where \( \mathcal{C} \) is a cartesian category, \((M, \eta, \mu, \iota) \) is a strong monad over \( \mathcal{C} \), and \( \theta \) is a map from the set of basic types to \( \text{Obj}(\mathcal{C}) \). We define \( [\tau]_{\text{am}}^{\Theta} \), the semantics of a type \( \tau \) under \( \Theta \). For simplicity, we usually write \([\tau]\) for \( [\tau]_{\text{am}}^{\Theta} \). It is defined by the following inductive definition:

\[ [A] = \theta(A) \quad (A \text{ is a basic type}), \quad [M\tau] = M[[\tau]], \quad [1] = 1, \quad [\tau_1 \times \tau_2] = [[\tau_1]] \times [[\tau_2]]. \]

For a context \( \Gamma = x_1 : \tau_1, \ldots, x_n : \tau_n \), we define \([\Gamma]\) as \([\tau_1] \times \cdots \times [\tau_n]\).

**Semantics of terms:** For a term \( e \) for which \( \Gamma \vdash e : \tau \) is derivable, its interpretation \([\Gamma \vdash e : \tau]\) is defined as a morphism from \([\Gamma]\) to \([\tau]\).

First, we choose a map \( \phi \) which maps each function symbol \( f \) of arity \( \tau_1 \to \tau_2 \) to a morphism \( \phi(f) \) from \([\tau_1]\) to \([\tau_2]\). Then \([\Gamma \vdash e : \tau]_{\text{am}}^I \) is defined by induction on the derivation of \( \Gamma \vdash e : \tau \), where \( I \) is the pair \((\Theta, \phi)\). We usually abbreviate \([\Gamma \vdash e : \tau]_{\text{am}}^I \) as \([\Gamma \vdash e : \tau]\).

Variables, *, pairing and projections are interpreted as usual. Function call \( \Gamma \vdash f(e_1) : \tau_2 \) \((f : \tau_1 \to \tau_2)\) is interpreted by \([\Gamma \vdash e_1 : \tau_1]; \phi(f)\). \([\Gamma \vdash [e] : M\tau]\) is defined as \([\Gamma \vdash e : [\tau]]_{\eta[I]}\).

Interpretation of let-expressions is the key point of Moggi's semantics. Suppose

\[ [\Gamma \vdash e_1 : M\tau_1] = g_1 : [[\Gamma]] \to M[[\tau_1]], \]

\[ [\Gamma, x_1 : \tau_1 \vdash e_2 : M\tau_2] = g_2 : [[\Gamma]] \times [[\tau_1]] \to M[[\tau_2]]. \]

At first sight, the definition

\[ [\Gamma \vdash (\text{let } x_1 \leftarrow e_1 \text{ in } e_2) : M\tau_2] = \langle \text{id}_{[[\Gamma]]}, g_1 \rangle ; g_2^* : [[\Gamma]] \to M[[\tau_2]] \]

seems to work. However, we cannot compose \( \langle \text{id}_{[[\Gamma]]}, g_1 \rangle \) and \( g_2^* \), since the codomain of \( \langle \text{id}_{[[\Gamma]]}, g_1 \rangle \) is \([\Gamma] \times M[[\tau_1]]\), while the domain of \( g_2^* \) is \( M([\Gamma] \times [\tau_1]) \). Accordingly,
Table 1
Summary of Moggi’s semantics

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \vdash \tau_i \text{ type} ) ((1 \leq i \leq n))</td>
<td>( \pi_i )</td>
</tr>
<tr>
<td>( x_1 : \tau_1, \ldots, x_n : \tau_n \vdash x_j : \tau_j )</td>
<td>( \Rightarrow \pi_i )</td>
</tr>
<tr>
<td>( \Gamma \vdash \ast : \Gamma )</td>
<td>( \Rightarrow \Gamma )</td>
</tr>
<tr>
<td>( \Gamma \vdash e_1 : \tau_1 )</td>
<td>( \Rightarrow g_1 )</td>
</tr>
<tr>
<td>( \Gamma \vdash e_2 : \tau_2 )</td>
<td>( \Rightarrow g_2 )</td>
</tr>
<tr>
<td>( \Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2 )</td>
<td>( \Rightarrow (g_1, g_2) )</td>
</tr>
<tr>
<td>( \Gamma \vdash e : \tau_1 \times \tau_2 )</td>
<td>( \Rightarrow g )</td>
</tr>
<tr>
<td>( \Gamma \vdash \pi_i(e) : \tau_i )</td>
<td>( \Rightarrow g; \pi_i )</td>
</tr>
<tr>
<td>( f : \tau_1 \rightarrow \tau_2 ) (function symbol)</td>
<td>( \Rightarrow g )</td>
</tr>
<tr>
<td>( \Gamma \vdash e_1 : \tau_1 )</td>
<td>( \Rightarrow g; \phi(f) )</td>
</tr>
<tr>
<td>( \Gamma \vdash e : \tau )</td>
<td>( \Rightarrow g )</td>
</tr>
<tr>
<td>( \Gamma \vdash [e] : M\tau )</td>
<td>( \Rightarrow g; \eta_{\tau_1} )</td>
</tr>
<tr>
<td>( \Gamma \vdash e_1 : M\tau_1 )</td>
<td>( \Rightarrow g_1 )</td>
</tr>
<tr>
<td>( \Gamma, x_1 : \tau_1 \vdash e_2 : M\tau_2 )</td>
<td>( \Rightarrow g_2 )</td>
</tr>
<tr>
<td>( \Gamma \vdash (\text{let } x_1 \leftarrow e_1 \text{ in } e_2) : M\tau_2 )</td>
<td>( \Rightarrow (id_{M\tau_1}; \eta_{\tau_1}) ; g_2^\ast )</td>
</tr>
</tbody>
</table>

we need the tensorial strength here. Since \( t_{\Gamma\mid\tau_1} \) is a morphism from \([\Gamma] \times M[\tau_1]\) to \(M([\Gamma] \times [\tau_1])\), we can compose \( (id_{M\tau_1}; g_1) \) and \( t_{\Gamma\mid\tau_1}; g_2^\ast \). Consequently, we define

\[
[\Gamma \vdash (\text{let } x_1 \leftarrow e_1 \text{ in } e_2) : M\tau_2] = (id_{M\tau_1}; \eta_{\tau_1}) ; g_2^\ast : [\Gamma] \rightarrow M[\tau_2].
\]

This semantics is summarized in Table 1.

2.3. Formal system for equational reasoning

Moggi [7] proposed a formal system for judging equality of terms. It has judgments of the form \( \Gamma \vdash e_1 =_\tau e_2 \), whose intuitive meaning is that \( e_1 \) and \( e_2 \) are equal as elements of \( \tau \) under the typing context \( \Gamma \). The inference rules of his system are summarized in Table 2. We write \([e/x]\phi\) for the result of substitution of \( e \) for the free occurrences of \( x \) in \( \phi \). The interpretation of judgments is as follows: \( \Gamma \vdash e_1 =_\tau e_2 \) is true (under \( I \)) if and only if \( [\Gamma] \vdash e_1 : \tau_{\text{def}} = [\Gamma] \vdash e_2 : \tau_{\text{def}} \) holds. Let \( E \) be an arbitrary set of equality judgments; then, by definition, \( I \) is a model of \( E \) if and only if every judgment in \( E \) is true under \( I \). Moggi proved the following theorem:

Theorem 2.1 (Moggi [7]). This formal system is sound and complete with respect to the semantics given in Section 2.2.

Proof. See [7]. \( \Box \)
Table 2

Equality rules for the metalanguage terms

<table>
<thead>
<tr>
<th>Inference rules of many sorted equational logic:</th>
</tr>
</thead>
<tbody>
<tr>
<td>refl ( \Gamma \vdash e : e )</td>
</tr>
<tr>
<td>symm ( \Gamma \vdash e_1 = e_2 )</td>
</tr>
<tr>
<td>trans ( \Gamma \vdash e_1 = e_2, \Gamma \vdash e_2 = e_3 )</td>
</tr>
<tr>
<td>subst ( \Gamma \vdash e : e ), ( \Gamma, x : e \vdash \phi )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rules for product types:</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \eta ) ( \Gamma \vdash e : 1 )</td>
</tr>
<tr>
<td>( \xi ) ( \Gamma \vdash x : 1 )</td>
</tr>
<tr>
<td>( \times \beta ) ( \Gamma \vdash (e_1, e_2) = \tau )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rules for computational types:</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \bot ) ( \Gamma \vdash e : \tau )</td>
</tr>
<tr>
<td>( \bot_\beta ) ( \Gamma \vdash \lambda x : \tau. e : \tau' )</td>
</tr>
<tr>
<td>( \bot ) ( \Gamma \vdash (e_1 \downarrow e_2) = \tau )</td>
</tr>
<tr>
<td>( \bot_\mathrm{ass} ) ( \Gamma \vdash \lambda x : \tau. e : \tau' )</td>
</tr>
</tbody>
</table>

3. Extended metalanguage

Here we introduce an extended version of the computational metalanguage which have function types \( \tau_1 \to \tau_2 \), weak initial type \( 0 \) and weak coproduct types \( \tau_1 + \tau_2 \). We will need this version in Section 7 to define a realizability interpretation.

First we add the following three new rules to the type-formation rules: (5) \( 0 \) is a type, (6–7) if \( \tau_1 \) and \( \tau_2 \) are types, then \( \tau_1 + \tau_2 \) and \( \tau_1 \to \tau_2 \) are types. Second we permit the language to have constant symbols. Recall that Moggi's original metalanguage has function symbols instead of constant symbols. Each constant symbol has its type. Of course, each constant symbol \( c \) of type \( \tau \) can be replaced by a term \( f_c(\ast) \) using a function symbol \( f_c \) of arity \( 1 \to \tau \). Conversely, \( f(e) \) (\( f \) a function symbol) can be replaced by an application term \( c(f(e)) \) using a constant \( c \), since we have function types now. In the following we assume, for simplicity, that we have no function symbols. Third we add the following new term-formation rules:

**\( \Gamma \vdash c : \tau \) (\( c \) is a constant of type \( \tau \))**

**\( \Gamma \vdash e : 0 \vdash \tau \) type**

**\( \Gamma \vdash e \in \tau_\ast (e) : \tau \) type**

**\( \Gamma \vdash e : M0 \vdash \tau \) type**

**\( \Gamma \vdash ?_\tau(e) : \tau \) type**

**\( \Gamma, x : \tau_1 \vdash e : \tau_2 \) type**

**\( \Gamma \vdash (\lambda x : \tau_1. e) : \tau_1 \to \tau_2 \) type**

**\( \Gamma \vdash e_1 : \tau_1 \) type**

**\( \Gamma \vdash ee_1 : \tau_2 \) type**
Finally, we add the following rules to our formal system of equality judgments:

\[
\begin{align*}
\text{app. } &\frac{\Gamma \vdash e_1 : \tau_1 \vdash e_2 : \tau_2 \text{ type}}{\Gamma \vdash e_1 e_2 : \tau_1 + \tau_2} \\
\lambda. \ &\frac{\Gamma \vdash e : \tau_1 \rightarrow \tau_2 \text{ type}}{\Gamma \vdash \lambda x : \tau_1. e : \tau_2} \\
\text{inl. } &\frac{\Gamma \vdash e_1 : \tau_1 \vdash e_1' : \tau_2 \text{ type}}{\Gamma \vdash \text{inl}_{\tau_1, \tau_2}(e_1) : \tau_1 + \tau_2} \\
\text{inr. } &\frac{\Gamma \vdash e_2 : \tau_2 \vdash e_2' : \tau_1 \text{ type}}{\Gamma \vdash \text{inr}_{\tau_1, \tau_2}(e_2) : \tau_1 + \tau_2} \\
\text{case. } &\frac{\Gamma \vdash e : \tau_1 \vdash e' : \tau_2 \text{ type}}{\Gamma \vdash \text{case } e \text{ of } \text{inl}_{\tau_1, \tau_2}(x_1) \Rightarrow c || \text{inr}_{\tau_1, \tau_2}(x_2) \Rightarrow d} \\
&\quad = \tau_3 \text{ (case } e' \text{ of } \text{inl}_{\tau_1, \tau_2}(x_1) \Rightarrow c' || \text{inr}_{\tau_1, \tau_2}(x_2) \Rightarrow d')
\end{align*}
\]

where \(DV(\Gamma)\) is the set of variables declared in \(\Gamma\).

Semantics: Suppose that \(\mathcal{C}\) is Cartesian closed and weakly co-Cartesian, and let \((M, \eta, \mu, 1)\) be a strong monad over \(\mathcal{C}\). Further, we assume that the monad functor \(M\) preserves weak initials; i.e. if 0 is a weak initial, then \(M0\) is also a weak initial. We write \(?A\) (\(\Theta_A\), respectively) for a specified weak initial arrow from 0 to \(A\) (from \(M0\) to \(A\), respectively). Since we may have constants, we must choose a map \(\gamma\) which maps each constant symbol \(c\) of type \(\tau\) to a global element \(\gamma(c) : 1 \rightarrow [\tau]\). The interpretation of a term is written as \([\Gamma \vdash e : \tau]^\Theta_{\mathcal{C}, \gamma}\), rather than \([\Gamma \vdash e : \tau]^\Theta_{\mathcal{C}, \phi, \gamma}\), because the interpretation \(\phi\) of function symbols is now not necessary. Usually, we omit the superscript \((\Theta, \gamma)\). Then the semantics of new types and terms are defined as shown in Table 3.

Here we make some remarks on weak initials. The only weak initial object of \(\mathbf{Set}\) is the empty set. The non-deterministic side effect monad over \(\mathbf{Set}\) defined in Section 2 does not preserve weak initials. However, the deterministic version over \(\mathbf{Set}\) preserves weak initial. In \(\mathbf{CPO}\), every object is weakly initial, and hence any strong monad over \(\mathbf{CPO}\) preserves weak initials.
Table 3
Semantics of the extended metalanguage

Semantics of types:

\[ [\tau_1 \rightarrow \tau_2] = [\tau_1] \Rightarrow [\tau_2] \] (exponential object)
\[ [0] = 0 \] (weak initial object)
\[ [\tau_1 + \tau_2] = [\tau_1] + [\tau_2] \] (weak coproduct)

Semantics of terms:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Gamma \vdash c : \tau )</td>
<td>( !_{[\tau]}; \gamma(c) )</td>
</tr>
<tr>
<td>( \vdash \tau ) type</td>
<td>( g )</td>
</tr>
<tr>
<td>( \Gamma \vdash \eta_{\tau}(e) : \tau )</td>
<td>( g; \eta_{[\tau]} )</td>
</tr>
<tr>
<td>( \vdash \tau ) type</td>
<td>( g )</td>
</tr>
<tr>
<td>( \Gamma \vdash e : M0 )</td>
<td>( g; \eta_M )</td>
</tr>
<tr>
<td>( \vdash \tau_2 ) type</td>
<td>( g )</td>
</tr>
<tr>
<td>( \Gamma \vdash \text{inl}_{\tau_1,\tau_2}(e_1) )</td>
<td>( g; \text{inl}_{[\tau_1]M[\tau_2]} )</td>
</tr>
<tr>
<td>( \vdash \tau_1 ) type</td>
<td>( g )</td>
</tr>
<tr>
<td>( \Gamma \vdash \text{inr}_{\tau_1,\tau_2}(e_2) )</td>
<td>( g; \text{inr}_{[\tau_1]M[\tau_2]} )</td>
</tr>
<tr>
<td>( \vdash e : \tau_1 + \tau_2 )</td>
<td>( g )</td>
</tr>
<tr>
<td>( \Gamma, x_1 : \tau_1, x_2 : \tau_2 \vdash d : \tau_3 )</td>
<td>( h_1 )</td>
</tr>
<tr>
<td>( \Gamma, x_1 : \tau_1, x_2 : \tau_2 \vdash d : \tau_3 )</td>
<td>( h_2 )</td>
</tr>
<tr>
<td>( \Gamma \vdash \text{case of inl}<em>{\tau_1,\tau_2}(x_1) \Rightarrow c | \text{inr}</em>{\tau_1,\tau_2}(x_2) \Rightarrow d : \tau_3 )</td>
<td>( (g; [A'h_1, A'h_2]), \eta_{[\Gamma]}); \text{eval} )</td>
</tr>
<tr>
<td>( \Gamma, x : \tau_1 \vdash e : \tau_2 )</td>
<td>( g )</td>
</tr>
<tr>
<td>( \Gamma \vdash (\lambda x : \tau_1.e) : \tau_1 \rightarrow \tau_2 )</td>
<td>( \Lambda g )</td>
</tr>
<tr>
<td>( \Gamma \vdash e : \tau_1 \rightarrow \tau_2 )</td>
<td>( g )</td>
</tr>
<tr>
<td>( \Gamma \vdash e_1 : \tau_1 )</td>
<td>( g_1 )</td>
</tr>
<tr>
<td>( \Gamma \vdash ee_1 : \tau_2 )</td>
<td>( (g, g_1); \text{eval} )</td>
</tr>
</tbody>
</table>

In the above, \( A'h_1 : [\tau_1] \rightarrow [\Gamma] \Rightarrow [\tau_3] \) is defined as \( A((\tau_2, \pi_1); h_1) \) (similarly for \( A'h_2 \))

4. Problems

Moggi’s semantics was successful and his type system is well suited for imperative functional programs. Then a natural question arises: “Can it be considered as a logical framework? Many type systems such as CoC (calculus of constructions) or Girard’s F can naturally encode constructive logics. Then how about Moggi’s system?” Actually, \( \eta_A : A \rightarrow MA \) looks like an inference rule “from \( A \), infer \( \bigcirc A \),” and \( \mu_A : M^2A \rightarrow MA \) looks like “from \( \bigcirc^2 A \), infer \( \bigcirc A \).” These two rules are derivable in S4 modal logic. Does Moggi’s type system correspond to any S4-like natural modal logic?

The answer is no. His computational metalanguage has, for each type \( A \) and \( B \), a term \( A \times MB \vdash e : M(A \times B) \) corresponding to tensorial strength \( t_{A,B} \). This means that
an unnatural inference rule "from $A \land \diamond B$, infer $\diamond (A \land B)$" is valid in his type system. Here is an excerpt from Moggi [7]:

...The semantics of computations corroborates the view that (constructive) proofs and programs are rather unrelated, although both of them can be understood in terms of functions. Indeed, monads (and comonads) used to model logical modalities, e.g., possibility and necessity in modal logic or why not and of course of linear logic, usually do not have a tensorial strength. In general, one should expect types suggested by logic to provide a more fine-grained type system without changing the nature of computations...

However, there is a gap in this argument. The fact that his system is complete for strong monad semantics does not imply that strength is absolutely necessary to interpret his system. Instead, we can generalize his semantics preserving soundness and completeness, avoiding use of strong monads. Our new semantics described in the next section is based on the notion of $L$-strong monad, in which tensorial strengths are replaced by what we call $L'$-strengths. Unlike tensorial strengths, $L'$-strengths correspond to the rule "from $\Box A \land \diamond B$, infer $\diamond (\Box A \land B)$." Fortunately, this rule is derivable in S4 modal logic. Moreover, $L'$-strong monads give sound and complete semantics of a constructive version of S4 modal logic called CS4 defined in Section 6. Since $\Box$ and $\diamond$ are not inter-definable in intuitionistic logic, CS4 has both modalities as primitive operators. An $L'$-strong monad has a comonad structure to model $\Box$-modality.

This result suggests that proofs and programs are closely related. In fact, we can show that proofs in our modal logic are considered as programs (see Section 7).

Then another question arises: if a proof in our modal logic is a program and a comonad is needed to interpret $\Box$-modality, why comonad types were not needed in Moggi's type system for programs? We shall answer this question in Section 9.

5. Generalized semantics of the metalanguage

In this section we generalize Moggi's semantics using the notion of $L'$-strong monad. In the following, we write $g^\land$ to mean the co-Kleisli lifting of $g$. When a comonad structure $\mathcal{L} = (L, \varepsilon, \delta)$ over a category $\mathcal{C}$ is given, we write $f;L g$ for the composition of $f$ and $g$ in the co-Kleisli category $\mathcal{C}^L$ induced by $\mathcal{L}$.

5.1. $L'$-strong monads

**Definition 5.1.** A cartesian comonad over a cartesian category $\mathcal{C}$ is a comonad which preserves the cartesian structure of $\mathcal{C}$. In detail, it is a comonad equipped with an isomorphism $m_1 : 1 \to L1$ and a natural isomorphism $m : L(-) \times L(-) \to L(- \times -)$ satisfying $m^{-1}_{A,B} = (L(\pi^A_B), L(n^A_B)) : L(A \times B) \to LA \times LB$. 
An important fact is that the co-Kleisli category $\mathcal{C}^L$ induced by a cartesian comonad is also cartesian. Its cartesian structure is given as follows:

produce $A \times^L B = A \times B$,

pairing $(f, g)^L = (f, g)$,

projection $\pi_i^L = e_{A_i} \times_{A_2}; \pi_i : L(A_1 \times A_2) \to A_i$,

terminal object $1^L = 1$,

terminal morphism $1_{A}^L = !_{LA} : LA \to 1$.

Note that $f \times^L g = m_{A,B}^{-1}(f \times g)$, where $f : LA \to C$ and $g : LB \to D$.

Moreover, if $\mathcal{C}$ is a ccc, $\mathcal{C}^L$ is also a ccc. Its exponential structure is given as follows:

exponential object $A =_{L} B = LA \Rightarrow B$,

currying $A^L f = A(m_{A,B}; f) : LA \Rightarrow B \Rightarrow^L C$ where $f : L(A \times B) \to C$,

evaluation $eval^L = (e_B \Rightarrow^L C \times^L id_{LB}); eval : L((B \Rightarrow^L C) \times^L B) \to C$.

**Definition 5.2.** Let $(L, e, \delta, m, m_1)$ be a cartesian comonad over $\mathcal{C}$ and $M$ be an endofunctor on $\mathcal{C}$. Consider the two functors $H, K : \mathcal{C}^L \times \mathcal{C} \to \mathcal{C}$ defined by the following:

$H(A, B) = LA \times MB$,

$K(A, B) = M(LA \times B)$ for objects $A, B$,

$H(f, g) = f^\wedge \times Mg$,

$K(f, g) = M(f^\wedge \times g)$ for morphisms $f, g$.

An $\mathcal{L}$-tensorial strength ($\mathcal{L}$-strength for short) for $M$ is, by definition, a natural transformation $\tau^L$ from $H$ to $K$ such that the following diagrams commute:

\[
\begin{align*}
M(A) & \xrightarrow{\tau^L_{AB}} M(LA \times A) \\
L1 \times MA & \xrightarrow{\tau^L_{LA}} M(L1 \times A) \\
L(A \times B) \times MC & \xrightarrow{\tau^L_{ABC}} M(L(A \times B) \times C) \\
LA \times (LB \times MC) & \xrightarrow{id_{LA} \times \tau^L_{BC}} LA \times M(LB \times C) \\
& \xrightarrow{\tau^L_{ABC}} M(LA \times (LB \times C))
\end{align*}
\]
where $r^L$ and $\alpha^L$ are natural isomorphisms
\[
\begin{align*}
r^L_A &= \pi^{L_1,A}_A : L_1 \times A \to A, \\
\alpha^L_{A,B,C} &= (m^{-1}_{A,B} \times \text{id}_C) ; \alpha_{LA, LB, LC} : L(A \times B) \times C \to LA \times (LB \times C).
\end{align*}
\]

**Definition 5.3.** An $\mathcal{L}$-strong monad over a cartesian category $\mathcal{C}$ is a triple $(\mathcal{M}, \mathcal{L}, t^L)$ satisfying the following conditions:
1. $\mathcal{L} = (L, \varepsilon, \delta, m, m_1)$ is a cartesian comonad over $\mathcal{C}$,
2. $\mathcal{M} = (M, \eta, \mu)$ is a monad over $\mathcal{C}$,
3. $t^L$ is a $\mathcal{L}$-tensorial strength for $M$,
4. the following diagrams commute:

Note that a strong monad is a special case of $\mathcal{L}$-strong monad of which comonad structure is trivial. As is easily seen, a tensorial strength for $M$ automatically becomes a $\mathcal{L}$-tensorial strength for $M$ for any $\mathcal{L}$. The converse is not true in general as we shall see later.

**Example.** We show an example of $\mathcal{L}$-strong monad over $\text{CPO}$. First we define a cartesian comonad structure $\mathcal{L}_1 = (L_1, \varepsilon, \delta, m, m_1)$ as follows:
- For a cpo $A = (A, \sqsubseteq)$, let $L_1A$ be the set of infinite increasing sequences over $A$; i.e., the underlying set of $L_1A$ is $\{ (a_n)_{n=0}^{\infty} \mid a_n \sqsubseteq a_{n+1} \text{ for all } n \geq 0 \}$. The elements are ordered componentwise; $(a_n)_{n=0}^{\infty} \sqsubseteq (a'_n)_{n=0}^{\infty}$ iff for all $n \geq 0$, $a_n \sqsubseteq a'_n$.
- For a morphism $f : A \to B$, let $L_1f : L_1A \to L_1B$ be defined by $(L_1f)(<a_n>_{n=0}^{\infty}) = <f(a_n)>_{n=0}^{\infty}$.
- $\varepsilon_A(<a_n>_{n=0}^{\infty}) = \bigsqcup_{n=0}^{\infty} a_n$.
- $\delta_A(<a_n>_{n=0}^{\infty}) = (s_n)_{n=0}^{\infty}$, where $(s_n)_i = \begin{cases} a_i & \text{if } i < n, \\ a_n & \text{otherwise.} \end{cases}$
- $m_{A,B}(<a_n>_{n=0}^{\infty}, <b_n>_{n=0}^{\infty}) = ((a_n, b_n))_{n=0}^{\infty}$.
- $m_1(\bot) = (\bot)_{n=0}^{\infty}$.

This comonad structure is the same as the "increasing paths" comonad $T_1$ studied in [2], except that $T_1$ is defined on bounded-complete algebraic cpo's. Next, let $(\mathcal{M}, \eta, \mu, t)$ be an arbitrary strong monad over $\text{CPO}$. Then $(\mathcal{L}_1, \mathcal{M}, t)$ becomes an $\mathcal{L}$-strong monad, where $\mathcal{M} = (M, \eta, \mu)$. 
Table 4
Summary of the \(L\)-strong monad semantics of the metalanguage

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\vdash \tau_i \text{ type } (1 \leq i \leq n))</td>
<td>(\pi^L_{\tau_i})</td>
</tr>
<tr>
<td>(x_1 : \tau_1, \ldots, x_n : \tau_n \vdash x_i : \tau_i)</td>
<td>({L}_{[\tau_i]})</td>
</tr>
<tr>
<td>(\Gamma \vdash \ast : 1)</td>
<td>(g_1)</td>
</tr>
<tr>
<td>(\Gamma \vdash e_1 : \tau_1)</td>
<td>(g_2)</td>
</tr>
<tr>
<td>(\Gamma \vdash e_2 : \tau_2)</td>
<td>(\langle g_1, g_2 \rangle^L)</td>
</tr>
<tr>
<td>(\Gamma \vdash \langle e_1, e_2 \rangle : \tau_1 \times \tau_2)</td>
<td>(g)</td>
</tr>
<tr>
<td>(\Gamma \vdash e : \tau_1 \times \tau_2)</td>
<td>(g^L \pi^L_{\tau_i})</td>
</tr>
<tr>
<td>(\Gamma \vdash \pi(e) : \tau_i)</td>
<td>(\langle id_{\tau_1}, g_1 \rangle; \tau_1; \eta_{\tau_1};\langle m_1, \tau_1, \tau_1; g_2 \rangle^*)</td>
</tr>
</tbody>
</table>

The monad structure of the above example has a tensorial strength. In Sections 6.5 and 7.5, we will find examples from logic of which monad structure has no tensorial strength.

5.2. \(L\)-strong monad semantics

Now, we shall define a new semantics of the computational metalanguage.

Semantics of types: The interpretation of types is given by \(\Theta = (\mathcal{C}, (\mathcal{M}, \mathcal{L}, t^L), \theta)\), where \(\mathcal{C}\) is a cartesian category, \((\mathcal{M}, \mathcal{L}, t^L)\) is an \(L\)-strong monad structure over \(\mathcal{C}\), and \(\theta\) is a map from the set of basic types to \(\text{Obj}(\mathcal{C})\).

We define \([\tau]_{\text{ml-L}}^\Theta\), which is usually written as \([\tau]\). Our interpretation of types is the same as Moggi's one except that \([M\tau]\) is defined as \(ML[\tau]\).

Semantics of terms: The semantics \([\Gamma \vdash e : \tau]_{\text{ml-L}}^\Theta\) is defined as a morphism from \(L[\Gamma]\) to \([\tau]\), i.e. it is defined as a morphism from \([\Gamma]\) to \([\tau]\) in the co-Kleisli category \(\mathcal{C}^L\). Here \(I\) is a pair \((\Theta, \varphi)\), and \(\varphi\) is a map which maps each function symbol \(f\) of arity \(\tau_1 \rightarrow \tau_2\) to a morphism \(\varphi(f) : L[\tau_1] \rightarrow [\tau_2]\). \(I\) is called an interpretation. We usually write \([\Gamma \vdash e : \tau]\) for \([\Gamma \vdash e : \tau]_{\text{ml-L}}^\Theta\). Our interpretation is defined so that it will coincide with Moggi's one when the comonad structure is trivial.

The interpretation of variables, \(\ast\), pairing, projections and function calls \(f(e)\) is easily defined (see Table 4), because \(\mathcal{C}^L\) is cartesian. \([\Gamma \vdash e : M\tau]\) is defined as \([\Gamma \vdash e : \tau]_{\text{ml-L}}^\Theta\). Let-expressions are interpreted as follows: Suppose

\[
[\Gamma \vdash e_1 : M\tau_1] = g_1 : L[\Gamma] \rightarrow ML[\tau_1],
\]
\[
[\Gamma, x_1 : \tau_1 \vdash e_2 : M\tau_2] = g_2 : L([\Gamma] \times [\tau_1]) \rightarrow ML[\tau_2].
\]
Then we have
\[ (m_{[\Gamma];[\tau_1]}, g_2) : L[\Gamma] \times L[\tau_1] \to ML[\tau_2], \]
\[ (m_{[\Gamma];[\tau_1]}, g_2)^* : M(L[\Gamma] \times L[\tau_1]) \to ML[\tau_2], \]
and
\[ t_{[\Gamma],L[\tau_1]}^L : L[\Gamma] \times ML[\tau_1] \to M(L[\Gamma] \times L[\tau_1]). \]

Using these morphisms, we define
\[ [\Gamma \vdash (\text{let } x_1 \leftarrow e_1 \text{ in } e_2) : M\tau_2] \]
\[ = (id_{L[\Gamma]}, g_1) ; t_{[\Gamma],L[\tau_1]}^L ; (m_{[\Gamma],[\tau_1]}, g_2)^* : L[\Gamma] \to ML[\tau_2]. \]

Note that the \( \mathcal{L} \)-tensorial strength is used here instead of a tensorial strength.

**Interpretation of equality judgments:** We say that a judgment \( \Gamma \vdash e_1 =_\tau e_2 \) is true under \( I \) if and only if \( [\Gamma \vdash e_1 : \tau]^{\mathcal{L}_{\text{ml}}}_{\mathcal{L}_{\text{ml}}} = [\Gamma \vdash e_2 : \tau]^{\mathcal{L}_{\text{ml}}}_{\mathcal{L}_{\text{ml}}} \) holds. Let \( E \) be an arbitrary set of equality judgments; then, by definition, \( I \) is a model of \( E \) if and only if every judgment in \( E \) is true under \( I \). Then we have the following theorem:

**Theorem 5.4.** The formal system given in Table 2 is sound and complete for the above interpretation. Speaking more exactly:

- **(Soundness)** Let \( E \) be an arbitrary set of equality judgments, and \( I \) be any model of \( E \). Then every judgment derived from \( E \) in the formal system is true under \( I \).
- **(Completeness)** Let \( E \) be an arbitrary set of equality judgments. If an equality judgment \( J \) is true in every model \( I \) of \( E \), then \( J \) is derived from \( E \) in the formal system.

**Proof.** Completeness is proved very easily using Theorem 2.1. Recall that Moggi's semantics is a special case of our new semantics. Suppose that \( J \) is true in every model \( I \) of \( E \), where \( I \) ranges over all interpretations in all \( \mathcal{L} \)-strong monad structures. Then, in particular, \( J \) is true in every model \( I' \) of \( E \) in the sense of Moggi's semantics. Hence, by Theorem 2.1, \( J \) is derived from \( E \).

Soundness is proved by tedious but straightforward calculations of arrows. It is sufficient to show that each inference rule preserves truth. For illustration of the proof, we prove the case of \( M\cdot \beta \) rule. The other cases are similar. Let \( A = [\tau_1], B = [\tau_2], C = [\Gamma], g_1 = [\Gamma \vdash e_1 : \tau_1] \) and \( g_2 = [\Gamma, x_1 : \tau_1 \vdash e_2 : M\tau_2] \). Then,
\[
[\Gamma \vdash \text{let } x_1 \leftarrow [e_1] \text{ in } e_2 : M\tau_2] \\
= (id_{L[\Gamma]}, (g_1^\wedge; \eta_{L\cdot A})); t_{C,L\cdot A}^{L\cdot A}; (m_{C,A}; g_2)^* \\
= (id_{L[\Gamma]}, (g_1^\wedge); (id_{L\cdot C} \times \eta_{L\cdot A})); t_{C,L\cdot A}^{L\cdot A}; (m_{C,A}; g_2)^* \\
= (id_{L[\Gamma]}, (g_1^\wedge); \eta_{L\cdot C \times L\cdot A} + (m_{C,A}; g_2)^* \\
= (id_{L[\Gamma]}, (g_1^\wedge); m_{C,A}; g_2) \\
\]
6. Constructive modal logic

In this section, we consider a constructive propositional modal logic called CS4, which corresponds to the notion of a \( \mathcal{L} \)-strong monad.

6.1. CS4

CS4 has \( \land, \lor, \text{ and } \rightarrow \) as propositional connectives; \( \top \) (true) and \( \bot \) (false) as propositional constants; and \( \Box, \Diamond \) as modal operators. Of course, it has propositional variables. Its axioms and rules are as follows:

1. It has all the axioms and rules of intuitionistic propositional logic.
2. It has the necessitation rule for \( \Box \): If \( A \) is a theorem (i.e. provable with no assumptions), then \( \Box A \) is also a theorem.
3. It has the following seven axioms on the modal operators:

\[
\begin{align*}
\Box \neg A & : (A \rightarrow B) \rightarrow (\Box A \rightarrow \Box B) \\
\Diamond \neg A & : (A \rightarrow B) \rightarrow (\Diamond A \rightarrow \Diamond B) \\
\Box T & : \Box A \rightarrow A \\
\Diamond T & : A \rightarrow \Diamond A \\
\Box 4 & : \Box A \rightarrow \Box^2 A \\
\Diamond 4 & : \Diamond A \rightarrow \Diamond^2 A \\
\Diamond \bot E & : \Diamond \bot \rightarrow A.
\end{align*}
\]

We define negation \( \neg A \) as \( A \rightarrow \bot \).

Clearly, CS4 is a subsystem of the classical S4 modal logic. In classical S4, \( \Box \) and \( \Diamond \) are inter-definable. However, it is not possible in CS4. CS4 does not prove \( \neg \Box \neg A \rightarrow \Diamond A \).

When a theory \( U \) is given by adding some formulas to CS4 as extra axioms, we say that \( U \) is a theory based on CS4.

6.2. Type theoretical formulations

We shall consider a type theory called TCS4 which is an extension of the computational metalanguage. In later sections, we show that TCS4 exactly corresponds to CS4 by the Curry–Howard correspondence.
The type formation rules of TCS4 are similar to those of the extended metalanguage except that the following new type-formation rule is added:

\[ \vdash A \text{ type} \]
\[ \vdash LA \text{ type} \]

A type of the form \( LA \) is called a comonad type.

The type inference rules are also similar to those of the extended metalanguage except that: (1) two kinds of new terms

\[ \text{box } f \text{ with } e_1, \ldots, e_n \text{ for } x_1, \ldots, x_n \]

and

\[ \text{unbox } e \]

are introduced and the type inference rules for them are added; and (2) the syntax of a let-expression is changed to

\[ \text{let } x \leftarrow e \text{ in } f \text{ with } e_1, \ldots, e_n \text{ for } x_1, \ldots, x_n \]

\((x_1, \ldots, x_n, x\) are variables, and \(e_1, \ldots, e_n\) are terms) and the type inference rule for it is also changed. Free variables of these new terms are determined as follows:

1. In \( d \models (\text{box } f \text{ with } e_1, \ldots, e_n \text{ for } x_1, \ldots, x_n) \), all free variables of \( f \) must be contained in \( \{x_1, \ldots, x_n\} \), and all of them become bound in \( d \). The free variables in \( e_1, \ldots, e_n \) remain free in \( d \).

2. The free variables of \( \text{unbox } e \) are those of \( e \).

3. In \( d \models (\text{let } x \leftarrow e \text{ in } f \text{ with } e_1, \ldots, e_n \text{ for } x_1, \ldots, x_n) \), all free variables of \( f \) must be in \( \{x_1, \ldots, x_n, x\} \), and all of them become bound in \( d \). The free variables in \( e_1, \ldots, e_n, e \) remain free in \( d \).

Notation 6.1. Sometimes we write

\[ L^n A \] for \( \overbrace{L \cdots L}^{n} A \) (\( M^n A \) is similar)

\( \tilde{e}_n \) for \( e_1, \ldots, e_n \),

\( \tilde{d}_{i \leq i < k} \) for \( d_1, \ldots, d_{k-1} \),

\( \tilde{d}_{k < i \leq m} \) for \( d_{k+1}, \ldots, d_m \),

\( \tilde{x}_n : \tilde{A}_n \) for \( x_1 : A_1, \ldots, x_n : A_n \),

\( \text{box } f \) for \( \text{box } f \) with \( e_1, \ldots, e_n \) for \( x_1, \ldots, x_n \) if \( n = 0 \),

\( \text{let } x \leftarrow e \) in \( f \) for \( \text{let } x \leftarrow e \) in \( f \) with \( e_1, \ldots, e_n \) for \( x_1, \ldots, x_n \) if \( n = 0 \).

Table 5 summarizes the inference rules of TCS4. The definition of TCS4 depends on the choice of the set of constant symbols. Accordingly, when a set \( C \) of constant
Table 5
Type inference rules of TCS4

Rules for constants and variables:

\[
\text{Const. } \Gamma \vdash c : A \quad (c \text{ is a constant of type } A)
\]

\[
\text{Var. } \frac{x_1 : A_1, \ldots, x_n : A_n \vdash t : A}{x_i : A_i}
\]

Rules for product types:

\[
1.1 \quad \frac{}{\Gamma \vdash \star : 1}
\]

\[
\times.1 \quad \frac{}{\Gamma \vdash e_1 : A_1, \Gamma \vdash e_2 : A_2 \vdash (e_1, e_2) : A_1 \times A_2}
\]

\[
\times.\exists \quad \frac{}{\Gamma \vdash e : A_1 \times A_2 \vdash \pi_1(e) : A_1}
\]

Rules for coproduct types:

\[
0.E \quad \frac{}{\Gamma \vdash e : 0 \vdash \lambda_e : A}
\]

\[
+J.1 \quad \frac{}{\Gamma \vdash e_1 : A_1, \Gamma \vdash e_2 : A_2 \vdash \text{type } \Gamma \vdash \text{inl}_{A_1, A_2}(e_1) : A_1 + A_2}
\]

\[
+J.2 \quad \frac{}{\Gamma \vdash e_2 : A_2 \vdash \text{type } \Gamma \vdash \text{inr}_{A_1, A_2}(e_2) : A_1 + A_2}
\]

\[
+E \quad \frac{}{\Gamma \vdash e : A_1 + A_2, \Gamma, x_1 : A_1 \vdash c : A_3, \Gamma, x_2 : A_2 \vdash d : A_3
\]

\[
\text{case } e \text{ of } \text{inl}_{A_1, A_2}(x_1) \Rightarrow \| \text{inr}_{A_1, A_2}(x_2) \Rightarrow \}
\]

Rules for function types:

\[
\rightarrow.1 \quad \frac{}{\Gamma, x : A_1 \vdash e : A_2 \vdash (\lambda x : A_1 . e) : A_1 \rightarrow A_2}
\]

\[
\rightarrow.E \quad \frac{}{\Gamma \vdash e : A_1 \rightarrow A_2, \Gamma \vdash e_1 : A_1 \vdash e[e_1] : A_2}
\]

Rules for comonad types:

\[
LJ \quad \frac{}{\Gamma \vdash e_1 : LA_1, \ldots, e_n : LA_n \vdash f : B \quad (d \equiv x_1 : LA_1, \ldots, x_n : LA_n)}
\]

\[
LE \quad \frac{}{\Gamma \vdash e : LA \vdash \text{unbox}_{A}(e) : A}
\]

Rules for monad types:

\[
MI \quad \frac{}{\Gamma \vdash e : A \vdash \text{box}_{A}(e) : MA}
\]

\[
MF \quad \frac{}{\Gamma \vdash e : MA, \Gamma \vdash e_1 : LA_1, \ldots, e_n : LA_n \vdash f : MB \quad (A \equiv x_1 : LA_1, \ldots, x_n : LA_n)}
\]

\[
M0.E \quad \frac{}{\Gamma \vdash e : M0 \vdash \rho_{A}(e) : A}
\]

symbols is given, we write TCS4(C) to denote the type theory determined by C. If T is of the form TCS4(C), T is called a type theory based on TCS4.

6.3. Encoding modal logic in type theory

We show how to encode a theory based on CS4 in a type theory based on TCS4. In the following, we assume that a bijection $BType : PV \rightarrow BT$ is given, where $PV$
is the set of propositional variables of CS4 and $BT$ is the set of basic types of TCS4. We define a mapping $Type$ from propositions of CS4 to types of TCS4 inductively as follows:

\[
\begin{align*}
Type(P) &= BT\text{ype}(P) \quad (P \text{ is a propositional variable}) \\
Type(\top) &= 1 \quad Type(\bot) = 0 \\
Type(A \land B) &= Type(A) \times Type(B) \\
Type(A \lor B) &= Type(A) + Type(B) \\
Type(A \rightarrow B) &= Type(A) \rightarrow Type(B) \\
Type(\Box A) &= L(\text{Type}(A)) \\
Type(\Diamond A) &= M(\text{Type}(A))
\end{align*}
\]

Clearly, $Type$ becomes a bijection and hence we write $Prop$ for the inverse of $Type$. For a typing judgment $\Gamma \vdash e : A$, we define

\[
\begin{align*}
Prop(\Gamma) &= Prop(A_1), \ldots, Prop(A_n) \\
Prop(\Gamma \vdash e : A) &= (Prop(\Gamma) \vdash Prop(A))
\end{align*}
\]

provided that $\Gamma$ is $\{ x_1 : A_1, \ldots, x_n : A_n \}$.

Suppose that we have a mapping $AxConst$ which maps each proposition $P$ of CS4 to a constant symbol $AxConst(P)$ of type $Type(P)$. When a theory $U$ based on CS4 is given, we let $C_U = \{ AxConst(P) | P \in Ax(U) \}$, where $Ax(U)$ is the set of proper axioms of $U$ (i.e. axioms of $U$ which is not in original CS4). We define a type theory $\mathcal{T}(U)$ by $\mathcal{T}(U) = TCS4(C_U)$.

Conversely, when a type theory $T = TCS4(C)$ is given, we define a propositional theory $\mathcal{P}(T)$ by $\mathcal{P}(T) = CS4 + \{ \text{Prop(typeof}(c)) | c \in C \}$, where $\text{typeof}(c)$ is the type of the constant symbol $c$.

Then we have the following theorem:

**Theorem 6.2.** Let $A_1, \ldots, A_n$ be a sequence of propositions and $x_1, \ldots, x_n$ be a sequence of distinct variables. If we have $A_1, \ldots, A_n \vdash B$ in a propositional theory $U$ based on CS4, then

\[
x_1 : Type(A_1), \ldots, x_n : Type(A_n) \vdash e : Type(B)
\]

is derivable in $\mathcal{T}(U)$ for some term $e$.

**Proof.** Clearly, the intuitionistic propositional logic is encoded in $\mathcal{T}(U)$, since we have product, (weak) coproduct, and function types. Therefore, it is sufficient to consider the proper axioms, the necessitation rule, and the axioms for modal operators. We write $A'$ for $Type(A)$.

- **Proper axioms:** If $A$ is a proper axiom, we can prove $\vdash AxConst(A) : Type(A)$ in $\mathcal{T}(U)$.
- **Necessitation rule:** Suppose that $\vdash e : A'$ is derived for some $e$. Then we have $\vdash \Box e : LA'$.
• **Axiom** □K : □(A → B) → (□A → □B). We have

\[ \vdash \lambda x : L(A' \rightarrow B') . \lambda y : LA'. \text{box (unbox } x'\text{)(unbox } y') \text{ with } x, y \text{ for } x', y' \]

\[ : L(A' \rightarrow B') \rightarrow LA' \rightarrow LB' \]

in TCS4.

• **Axiom** □T : □A → A. We can derive \( \vdash \lambda x : LA'. \text{unbox } x : LA' \rightarrow A' \).

• **Axioms** □4 : □A → □²A. TCS4 proves \( \vdash \lambda x : LA'. \text{box } y \text{ with } x \text{ for } y : LA' \rightarrow L²A' \).

• **Axiom** □K : □(A → B) → (□A → □B). We have

\[ \vdash \lambda x : L(A' \rightarrow B') . \lambda y : MA'. \text{ let } y' \leftarrow y \text{ in } [(\text{unbox } x')(y')] \text{ with } x \text{ for } x' \]

\[ : I(A' \rightarrow B') \rightarrow MA' \rightarrow MB' \]

• **Axiom** □T : A → □A. We have \( \vdash \lambda x : A'. [x] : A' \rightarrow MA' \).

• **Axioms** □4 : □²A → □A. We can prove \( \vdash \lambda x : M²A'. \text{let } y \leftarrow x \text{ in } y : M²A' \rightarrow MA' \).

• **Axiom** □⊥E : □⊥ → A. We have \( \vdash \lambda x : M0.2M(x) : M0 \rightarrow A' \). \( \Box \)

Conversely, we have the following theorem:

**Theorem 6.3.** If \( \Gamma \vdash e : A \) is derived in type theory \( T \) based on TCS4, then we have \( \text{Prop}(\Gamma \vdash e : A) \) in \( \mathcal{PT}(T) \).

**Proof.** We prove by induction on the derivation of the typing judgment. Consider the last rule applied. Each rule for variables, product types, coproduct types, and function types clearly corresponds to an inference of the intuitionistic propositional logic. Hence, we consider the other rules.

• **Rule Const.** If \( c \) is a constant of type \( A \) in \( T \), we have \( \text{Prop}(A) \) as an axiom of \( \mathcal{PT}(T) \). Therefore, we have \( \text{Prop}(\Gamma) \vdash \text{Prop}(A) \).

• **Rule L.I.** We show that \( \Gamma \vdash \square B \) is derivable from

\[ \Gamma \vdash \square A_i \quad (1 \leq i \leq n) \]  \hspace{1cm} (1)

and

\[ \square A_1, \ldots, \square A_n \vdash B. \]  \hspace{1cm} (2)

From (1) and □4, we have

\[ \Gamma \vdash \square A_i \quad (1 \leq i \leq n). \]  \hspace{1cm} (3)

Further, we have

\[ \vdash \square(\square A_1 \rightarrow (\square A_2 \rightarrow (\cdots (\square A_n \rightarrow B) \cdots) )) \]  \hspace{1cm} (4)

from (2), successively applying implication introduction and necessitation. Therefore we can obtain \( \Gamma \vdash \square B \) from (4) and (3), repeatedly using □K.

• **Rule L.E.** If \( \Gamma \vdash \square A \), then \( \Gamma \vdash A \) by □T.
• **Rule M.I.** If \( \Gamma \vdash A \), then \( \Gamma \vdash \circ A \) by \( \circ T \).
• **Rule M.E.** We show that \( \Gamma \vdash \circ B \) is derivable from

\[
\Gamma \vdash \circ A,
\]

\[
\Gamma \vdash \square A_i \quad (1 \leq i \leq n)
\]

and

\[
\square A_1, \ldots, \square A_n, \Gamma \vdash \circ B.
\]

Just like the case of the \( L.I \) rule, we can derive

\[
\Gamma \vdash \square (A \rightarrow \circ B)
\]

from (6) and (7). Then we have \( \circ B \) from (8) and (5) using \( \circ K \) and \( \circ 4 \).

• **Rule M0.E.** If \( \Gamma \vdash \bot \), we have \( \Gamma \vdash A \) by \( \circ \bot E \). \( \square \)

### 6.4. Categorical semantics of CS4

Let us give a categorical semantics of theories based on CS4 using the notion of an \( \mathcal{L} \)-strong monad.

**Definition 6.4.** \((\mathcal{E}, (\mathcal{M}, \mathcal{L}, t^L))\) is a CS4 structure if and only if the following conditions hold:

1. \( \mathcal{E} \) is a weakly co-cartesian ccc.
2. \((\mathcal{M}, \mathcal{L}, t^L)\) is an \( \mathcal{L} \)-strong monad over \( \mathcal{E} \).
3. The monad functor of \( \mathcal{M} \) preserves weak initial objects.

The last condition is needed to interpret \( \circ \bot E \).

Suppose that a CS4 structure \( \mathcal{S} = (\mathcal{E}, (\mathcal{M}, \mathcal{L}, t^L)) \) is given. The interpretation of formulas of CS4 is defined as follows: First, we arbitrarily choose a mapping \( \sigma : PV \rightarrow \text{Obj}(C) \). The pair \( I = (\mathcal{S}, \sigma) \) is called an interpretation. Then we define \([A]_{\text{CS4}} \in \text{Obj}(C)\) for each formula \( A \). We often write \([A]\) for \([A]_{\text{CS4}}\). \([A]\) is inductively defined as follows:

\[
[P] = \sigma(P) \quad (P \text{ is a propositional variable})
\]

\[
[\top] = 1 \quad [\bot] = 0
\]

\[
[A \land B] = [A] \times [B] \quad [A \lor B] = [A] + [B]
\]

\[
[A \rightarrow B] = ([A] \Rightarrow [B])
\]

\[
[\square A] = L[A] \quad [\diamond A] = M[A].
\]

For \( \Gamma = A_1, \ldots, A_n \), define \([\Gamma]_{\text{CS4}}^L = [A_1]_{\text{CS4}}^L \times \cdots \times [A_n]_{\text{CS4}}^L \). If \( n = 0 \), let \([\Gamma]_{\text{CS4}}^L = 1 \).

**Definition 6.5.** (1) We say that a formula \( A \) is valid under an interpretation \( I \) if and only if \([A]_{\text{CS4}}^I \) has a global element.
We say that \( \Gamma \vdash A \) is valid under an interpretation \( I \) if and only if there exists a morphism from \( [\Gamma]_{CS4} \) to \( [A]_{CS4} \).

(3) \( I \) is called a model of \( U \) if and only if every provable formula of \( U \) is valid under \( I \).

This semantics is sound and complete in the following sense:

**Theorem 6.6.** Suppose that \( U \) is a theory based on CS4. Then the following propositions hold:

1. (Soundness) If each proper axiom of \( U \) is valid under \( I \), then \( I \) is a model of \( U \).
2. (Completeness) If \( A \) is not provable in \( U \), then there exists a model \( I \) of \( U \) such that \( A \) is not valid under \( I \).

We prove this theorem later, using a categorical semantics of TCS4.

### 6.5. Categorical semantics of TCS4

We shall define a categorical semantics of TCS4.

**Semantics of Types:** Assume that a CS4 structure \( \mathcal{S} = \langle C, (\mathcal{M}, \mathcal{L}, t^\mathcal{L}) \rangle \) is given. We choose a mapping \( \theta \) from the set of basic types \( BT \) to \( \text{Obj}(C) \). Then we define \( [A]_{\text{TCS4}}^\Theta \) for each type \( A \), where \( \Theta \) is the pair \( (\mathcal{S}, \theta) \). We often abbreviate \( [A]_{\text{TCS4}}^\Theta \) as \( [A] \). It is defined as follows:

\[
\begin{align*}
[A] &= \theta(A) \quad (A \text{ is a basic type}) \\
[1] &= 1 \\
[A \times B] &= [A] \times [B] \\
[A + B] &= [A] + [B] \\
[A \rightarrow B] &= ([A] \Rightarrow [B]) \\
[L^A] &= L[A] \\
[M^A] &= M[A].
\end{align*}
\]

For a typing context \( \Gamma = x_1 : A_1, \ldots, x_n : A_n \), we define \( [\Gamma] = [A_1] \times \cdots \times [A_n] \).

**Semantics of Terms:** To define the semantics of terms, we choose a map \( \gamma \) which maps each constant symbol \( c \) of type \( A \) to a global element \( \gamma(c) : 1 \rightarrow [A] \). The pair \( (\Theta, \gamma) \) is called an interpretation.

We define

\[
[\Gamma \vdash e : A]_{\text{TCS4}}^{\Theta, \gamma} : [\Gamma] \rightarrow [A]
\]

by induction on the derivation of \( \Gamma \vdash e : A \). The interpretation of constants, variables, rules for product, coproduct, and function types is similar to the case of the extended metalanguage. The interpretation of comonad types and monad types are as follows:

Since \( L \) preserves the cartesian structure,

\[
\langle L(\pi_1^n), \ldots, L(\pi_n^n) \rangle : L(A_1 \times \cdots \times A_n) \rightarrow L(A_1) \times \cdots \times L(A_n)
\]
Table 6
Categorical semantics of TCS4

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Gamma \vdash e_i : LA_i ) ((1 \leq i \leq n))</td>
<td>( h_i )</td>
</tr>
<tr>
<td>( \bar{x}_n : \overline{LA}_n \vdash f : B )</td>
<td>( g )</td>
</tr>
<tr>
<td>( \Gamma \vdash \text{(box} f \text{ with } \bar{e}_n \text{ for } \bar{x}_n) : LB )</td>
<td>( \langle h_1, \ldots, h_n \rangle ; m_{[A_1]} \times \ldots \times [A_n] ; ((m_{[A_1]} \times \ldots \times [A_n])^{-1} ; g) \wedge )</td>
</tr>
<tr>
<td>( \Gamma \vdash e : LA )</td>
<td>( g )</td>
</tr>
<tr>
<td>( \Gamma \vdash \text{unbox} e : A )</td>
<td>( g ; e_{[A]} )</td>
</tr>
<tr>
<td>( \Gamma \vdash [e] : MA )</td>
<td>( g )</td>
</tr>
<tr>
<td>( \Gamma \vdash e : MA )</td>
<td>( g_{[A]} )</td>
</tr>
<tr>
<td>( \bar{x}_n : \overline{LA}_n, x : A \vdash f : MB )</td>
<td>( g_2 )</td>
</tr>
<tr>
<td>( \Gamma \vdash (\text{let} x \leftarrow e \text{ in} f \text{ with } \bar{e}_n \text{ for } \bar{x}_n) : MB )</td>
<td>( \langle \langle h_1, \ldots, h_n \rangle ; m_{[A_1]} \times \ldots \times [A_n] ; g_1 \rangle ); ( t_{A_1} \times \ldots \times A_n, A ; \left((m_{[A_1]} \times \ldots \times [A_n])^{-1} \times id_{[A]} \right) ; g_2 \rangle )</td>
</tr>
<tr>
<td>( \Gamma \vdash e : M0 )</td>
<td>( g )</td>
</tr>
<tr>
<td>( \Gamma \vdash M^M_A (e) : A )</td>
<td>( g ; e^M_{[A]} )</td>
</tr>
</tbody>
</table>

is an isomorphism. We write \( m_{A_1} \times \ldots \times A_n \) for its inverse. Then we define

\[
[\Gamma \vdash (\text{box} f_1 \text{ with } e_1, \ldots, e_n \text{ for } x_1, \ldots, x_n) : LB] \\
= \langle h_1, \ldots, h_n \rangle ; m_{[A_1]} \times \ldots \times [A_n] ; ((m_{[A_1]} \times \ldots \times [A_n])^{-1} ; g) \wedge \\
[\Gamma \vdash (\text{let} x \leftarrow e \text{ in} f_2 \text{ with } e_1, \ldots, e_n \text{ for } x_1, \ldots, x_n) : MB] \\
= \langle \langle h_1, \ldots, h_n \rangle ; m_{[A_1]} \times \ldots \times [A_n] ; g_1 \rangle ; t_{A_1} \times \ldots \times A_n, A ; \left((m_{[A_1]} \times \ldots \times [A_n])^{-1} \times id_{[A]} \right) ; g_2 \rangle 
\]

where

\[
g = [\bar{x}_n : \overline{LA}_n \vdash f_1 : B] \\
h_i = [\Gamma \vdash e_i : LA_i] \quad (1 \leq i \leq n) \\
g_1 = [\Gamma \vdash e : MA] \\
g_2 = [\bar{x}_n : \overline{LA}_n, x : A \vdash f_2 : MB].
\]

The interpretation of the other rules is easy; see Table 6.

**Equality Rules:** According to this semantics, we introduce a formal system of equality judgments for TCS4-terms.

The system has the inference rules of the many sorted equational logic, rules for product, coproduct and function types. They are the same as those of the extended metalanguage described in Section 3. In addition, the system has rules for comonad types and monad types. They are summarized in Table 7.

By definition, \( \Gamma \vdash e_1 = e_2 \) is true if and only if \( [\Gamma \vdash e_1 : \tau] = [\Gamma \vdash e_2 : \tau] \) holds. The definition of a model is similar to those in Sections 2.3 and 5.2.
Table 7
Equality rules for TCS4 terms

Inference rules of many sorted equational logic, rules for product, coproduct and function types:
- Same as those of extended metalanguage.

### Rules for comonad types:

**box. $\xi$**
\[
\Gamma \vdash e_1 : L_{A_1} (1 \leq i \leq n) \quad \bar{x}_n \downarrow A \vdash f : B \\
\Gamma \vdash (\text{box } f \text{ with } \bar{e}_n \text{ for } \bar{x}_n) =_{LB} (\text{box } f' \text{ with } \bar{e}_n' \text{ for } \bar{x}_n)
\]

**unbox. $\xi$**
\[
\Gamma \vdash e : L_{A} \quad \bar{x}_n \downarrow A \vdash \text{unbox } e =_{A} \text{unbox } e'
\]

**box. ass. $k$**
\[
\Gamma \vdash e_1 : L_{A_1} (1 \leq i \leq n) \quad \bar{x}_n \downarrow A \vdash f : B_k \\
\Gamma \vdash d_i : L_{B_i} (1 \leq i \leq m, i \neq k) \quad \bar{y}_m \downarrow L_{B_m} \vdash g : C \\
\Gamma \vdash (\text{box } g \text{ with } \bar{d}_1 \downarrow k \downarrow i \downarrow k_i \downarrow i \downarrow m, \text{box } f \text{ with } \bar{e}_n \text{ for } \bar{x}_n), \bar{d}_k \downarrow i \downarrow m \text{ for } \bar{y}_m)
\]

=\_LC
\[
(\text{box } (\text{box } f \text{ with } \bar{e}_n \text{ for } \bar{x}_n) \downarrow y_k \downarrow k \downarrow m) \\
\text{with } \bar{d}_1 \downarrow k \downarrow i \downarrow k_i \downarrow i \downarrow m \text{ for } \bar{y}_1 \downarrow k \downarrow i \downarrow k_i \downarrow i \downarrow m)
\]

L. $\beta$
\[
\Gamma \vdash e_1 : L_{A_1} (1 \leq i \leq n) \quad \bar{x}_n \downarrow L_{A_n} \vdash f : B \\
\Gamma \vdash \text{unbox } (\text{box } f \text{ with } \bar{e}_n \text{ for } \bar{x}_n) =_{B} [\bar{e}_n/\bar{x}_n] f'
\]

L. $\eta$
\[
\Gamma \vdash e_1 : L_{A_1} (1 \leq i \leq n) \quad \bar{x}_n \downarrow L_{A_n} \vdash f : LB \\
\Gamma \vdash (\text{box } (\text{unbox } f) \text{ with } \bar{e}_n \text{ for } \bar{x}_n) =_{LB} [\bar{e}_n/\bar{x}_n] f
\]

### Rules for monad types:

$[-], \bar{\xi}$
\[
\Gamma \vdash e =_{A} e' \\
\Gamma \vdash [e] =_{MA} [e']
\]

**let. $\xi$**
\[
\Gamma \vdash e =_{MA} e' \\
\Gamma \vdash e_1 =_{L_{A_1}} (1 \leq i \leq n) \quad \bar{x}_n \downarrow L_{A_n} \downarrow x : A \vdash f =_{MB} f' \\
\Gamma \vdash (\text{let } x \leftarrow e \text{ in } f \text{ with } \bar{e}_n \text{ for } \bar{x}_n) =_{MB} (\text{let } x \leftarrow e' \text{ in } f' \text{ with } \bar{e}_n' \text{ for } \bar{x}_n)
\]

**let. ass**
\[
\Gamma \vdash e : MA \\
\Gamma \vdash e_1 : L_{A_1} (1 \leq i \leq n) \\
\Gamma \vdash x : L_{A_n} \downarrow x : A \vdash f =_{MB} x : B \\
\Gamma \vdash d_i : L_{B_i} (1 \leq i \leq m, i \neq k) \quad \bar{y}_m \downarrow L_{B_m} \downarrow x : A \vdash g : MC \\
\Gamma \vdash (\text{let } y \leftarrow (\text{let } x \leftarrow e \text{ in } f \text{ with } \bar{e}_n \text{ for } \bar{x}_n), \text{in } g \text{ with } \bar{d}_m \text{ for } \bar{y}_m)
\]

=\_MC
\[
(\text{let } y \leftarrow e \text{ in } f \text{ with } \bar{e}_n \text{ for } \bar{x}_n) \text{ with } \bar{e}_n, \bar{d}_m \text{ for } \bar{x}_n, \bar{y}_m)
\]

**let. box. comm. $k$**
\[
\Gamma \vdash e : MA \\
\Gamma \vdash e_1 : L_{A_1} (1 \leq i \leq n) \\
\Gamma \vdash d_i : L_{B_i} (1 \leq i \leq m, i \neq k) \quad \bar{y}_m \downarrow L_{B_m} \downarrow x : A \vdash g : MC \\
\Gamma \vdash (\text{let } x \leftarrow e \text{ in } f \text{ with } \bar{e}_n \text{ for } \bar{x}_n, \text{with } \bar{d}_1 \downarrow k \downarrow k_i \downarrow i \downarrow m \text{ for } \bar{y}_1 \downarrow k \downarrow i \downarrow k_i \downarrow i \downarrow m)
\]

M. $\beta$
\[
\Gamma \vdash e : A \\
\Gamma \vdash e_1 : L_{A_1} (1 \leq i \leq n) \\
\Gamma \vdash d_i : L_{B_i} (1 \leq i \leq m, i \neq k) \quad \bar{y}_m \downarrow L_{B_m} \downarrow x : A \vdash f =_{MB} x : B \\
\Gamma \vdash (\text{let } x \leftarrow [e] \text{ in } f \text{ with } \bar{e}_n \text{ for } \bar{x}_n) =_{MB} [e, \bar{e}_n/\bar{x}_n] f
\]

M. $\eta$
\[
\Gamma \vdash e : MA \\
\Gamma \vdash e_1 : L_{A_1} (1 \leq i \leq n) \\
\Gamma \vdash (\text{let } x \leftarrow e \text{ in } [x] \text{ with } \bar{e}_n \text{ for } \bar{x}_n) =_{MA} e
\]
We can prove the following theorem:

**Theorem 6.7.** This formal system is sound and complete with respect to the above interpretation.

**Proof.** Soundness is proved by straightforward calculations of morphisms. For illustration, we show that $M. \eta$ rule is valid. Let $g_1 = [\Gamma \vdash e: MA]$ and $h_t = [\Gamma \vdash e_t : LA_t] \ (1 \leq i \leq n)$. Then

\[
\begin{align*}
[\Gamma \vdash (\text{let } x \leftarrow e \text{ in } [x] \text{ with } \bar{e}_n \text{ for } \bar{x}_n) : MA] &= (\langle h_1, \ldots, h_n; m_{[A_1] \times \cdots \times [A_n]} \rangle; g_1)^t_{A_1 \times \cdots \times A_n, A}; \\
&= ((m_{[A_1] \times \cdots \times [A_n]})^{-1} \times id_{[A]}); \pi_{n+1}^{\eta t}; \eta_{[A]}^t) \\
&= (\langle h_1, \ldots, h_n; m_{[A_1] \times \cdots \times [A_n]} \rangle; g_1)^{t_{A_1 \times \cdots \times A_n, A}}; (\pi_2; \eta_{[A]}^t) \\
&= (\langle h_1, \ldots, h_n; m_{[A_1] \times \cdots \times [A_n]} \rangle; g_1)^{t_{A_1 \times \cdots \times A_n, A}; M(\pi_2)} \\
&= (\langle h_1, \ldots, h_n; m_{[A_1] \times \cdots \times [A_n]} \rangle; g_1; \pi_2) \\
&= g_1.
\end{align*}
\]

Hence $M. \eta$ rule is valid.

The proof of completeness is similar to Moggi's proof for Theorem 2.1. Suppose that an arbitrary set of equality judgments $E$ is given. Then we prove that: if an equation $\Gamma \vdash e_1 =_E e_2$ is true in all models of $E$, then the equation is derived from $E$ in the formal system. We define a category $\mathcal{F}(E)$ by the following construction:

- Objects of $\mathcal{F}(E)$ are types.
- A morphism from $A$ to $B$ is an equivalence class $[x: A \vdash e: B]_\sim$ of typed terms with respect to the equivalence relation $\sim$ defined by

\[
(x: A \vdash e_1 : B)_\sim \sim (x: A \vdash e_2 : B) \text{ if and only if } (x: A \vdash e_1 =_E e_2) \text{ is derived from } E,
\]

provided that $(x: A \vdash e: B)$ is identified with $(y: A \vdash [y/x]e: B)$ when $y$ does not occur in $e$. Terms of the form $(x_1 : A_1, \ldots, x_n : A_n \vdash e : B)$ with $n \neq 1$ are not considered as morphisms.

- Composition is substitution, i.e.

\[
(x: A \vdash e_1 : B)_\sim; (y: B \vdash f : C)_\sim = (x: A \vdash [e/y]f : C)_\sim
\]

- The identity on $A$ is $[x: A \vdash x : A]_\sim$.

Then $\mathcal{F}(E)$ has a CS4 structure $\mathcal{S} = (\mathcal{C}, (\mathcal{M}, \mathcal{L}, t^L))$ defined as follows:

- $\mathcal{C} = \mathcal{F}(E)$.
- The structure of ccc is defined by

\[
\begin{align*}
\mathcal{A} &= [x: A \vdash \ast : 1]_\sim, \\
\mathcal{A} \times \mathcal{B} &= \mathcal{A} \times \mathcal{B}, \\
\pi_{A}^{A,B} &= [x: A \times B \vdash \pi_1(x): A]_\sim \text{ and } \pi_{B}^{A,B} = [x: A \times B \vdash \pi_2(x): B]_\sim, \\
\langle [x: C \vdash e: A]_\sim, [x: C \vdash f: B]_\sim \rangle &= [x: C \vdash \langle e, f \rangle : A \times B]_\sim,
\end{align*}
\]
- $A \Rightarrow B = (A \rightarrow B)$,
- $A \vdash [x : C \times A \vdash e : B] = [x : C \vdash \lambda y : A. [(x,y)\,/\,x]e : A \rightarrow B]$, 
- $\text{eval}_{A,B} = [x : (A \rightarrow B) \times A \vdash (\pi_1(x))(\pi_2(x)) : B]$. 

- The weakly co-cartesian structure is defined by 
- $0 = 0$ and $?_A = [x : 0 \vdash ?_A(x) : A]$, 
- $A + B = A + B$, 
- $\text{inl}_{A,B} = [x : A \vdash \text{inl}_{A,B}(x) : A + B]$ and $\text{inr}_{A,B} = [x : B \vdash \text{inr}_{A,B}(x) : A + B]$, 
- The weak coproduct arrow $[[x : A \vdash e : C] \Rightarrow [y : B \vdash f : C]]$ is 
  
$\Rightarrow \vdash (\text{case } z \text{ of } \text{inl}_{A,B}(x) \Rightarrow e \parallel \text{inr}_{A,B}(y) \Rightarrow f) : C$.

- The cartesian comonad structure is defined by 
- $L(A) = LA$, 
- $e_\eta = [x : LA \vdash \text{unbox } x : A]$, 
- $[(x : LA \vdash e : B)] = [x : LA \vdash (\text{box } e \text{ with } x \text{ for } x) : LB]$, 
- $m_1 = [x : 1 \vdash (\text{box } *) : L1]$, 
- $m_{A,B}$ is 
  
$[x : LA \times LB \vdash (\text{box } (\text{unbox } y_1, \text{unbox } y_2) \text{ with } \pi_1(x), \pi_2(x) \text{ for } y_1, y_2) : L(A \times B)]$. 

- The monad structure is defined by 
- $M(A) = MA$, 
- $\eta_A = [x : A \vdash [x : MA]]$, 
- $([x : A \vdash e : MB])^* = [y : MA \vdash (\text{let } x \leftarrow y \text{ in } e) : MB]$. 

- The $\mathcal{L}'$-tensorial strength for $M$ is defined by 
  
$i_{A,B}^L = [x : LA \times MB \vdash y \leftarrow \pi_2(x) \text{ in } [(z,y)]] \text{ with } \pi_1(x) \text{ for } z : M(LA \times B)$. 

### Proposition 6.8. The above construction gives a CS4 structure. 

**Proof.** See the appendix.

This CS4 structure is called the canonical CS4 structure for $E$. We interpret the language in this CS4 structure as follows: Define $\theta(A) = A$ for each basic type $A$. Let $\Theta = (\mathcal{S}, \theta)$. Then $[A]^{\Theta}_{\text{ITCS4}} = A$ holds for any type $A$. Define $\gamma(e) = [x : 1 \vdash e : \text{typeof}(c)]$. Then we have 

$[\Gamma \vdash e : A]^{\Theta, \gamma}_{\text{ITCS4}} = [\Gamma \vdash e : A]$. 

This interpretation is called the canonical interpretation of $E$. 

Now suppose that $\Gamma \vdash e_1 =_A e_2$ is true in any model of $E$. Then, as a special case, it is true under the canonical interpretation of $E$. Therefore, $[\Gamma \vdash e_1 : A] = [\Gamma \vdash e_2 : A]$ holds, and hence $\Gamma \vdash e_1 =_A e_2$ is derivable from $E$. This shows that our formal system is complete.
Thus, we have proved Theorem 6.7. □

Note that the monad structure defined in this proof cannot have a tensorial strength in general, because \( A \land \circ B \rightarrow \circ(A \land B) \) is not necessarily valid in CS4.

**Proof of Theorem 6.6.** As an application of the above theorem, we shall give a proof of Theorem 6.6.

(Soundness) Suppose that each proper axiom of \( U \) is valid under \( I = (\mathcal{S}, \sigma) \). Choose an element \( e_A : 1 \rightarrow [A]_{\text{CS4}} \) for each proper axiom \( A \). Then we construct an interpretation of \( \mathcal{T}(U) \) as follows: First define \( \theta(A) = [\text{Prop}(A)]_{\text{CS4}} \). Let \( \Theta = (\mathcal{S}, \theta) \). Clearly \( [\text{Type}(A)]_{\text{CS4}}^\Theta = [A]_{\text{CS4}} \). Then define \( \gamma \) by \( \gamma(c) = e_A \), where \( c = \text{AxConst}(A) \). Note that \( \gamma \) is well-defined, because each constant symbol of \( \mathcal{T}(U) \) is of the form \( \text{AxConst}(A) \) and of type \( \text{Type}(A) \) for some proper axiom \( A \) of \( U \).

Now let \( A \) be a provable formula of \( U \). Then, by Theorem 6.2, we have \( \vdash e : \text{Type}(A) \) in \( \mathcal{T}(U) \) for some \( e \). Therefore we have \( \vdash [\vdash e : \text{Type}(A)]_{\text{CS4}}^\Theta, \gamma : 1 \rightarrow [\text{Type}(A)]_{\text{CS4}}^\Theta \). This means that \( A \) is valid under \( I \), because \( [\text{Type}(A)]_{\text{CS4}}^\Theta = [A]_{\text{CS4}} \). Since \( A \) is arbitrary, \( I \) is a model of \( U \).

(Completeness) Suppose that \( A \) is not provable in \( U \). Then we have no \( e \) such that \( \vdash e : \text{Type}(A) \) in \( \mathcal{T}(U) \). Consider the canonical interpretation \( I = (\Theta, \gamma) = ((S, \theta, \gamma)) \) of \( E \), where \( E \) is the set of all provable equality judgments of \( \mathcal{T}(U) \). Then \( [\text{Type}(A)]_{\text{CS4}}^\Theta \) has no global element. This means that \( A \) is not valid under the model \( I \), where \( I = (\mathcal{S}, \sigma) \) is defined by \( \sigma(P) = \theta(\text{Type}(P)) \). □

### 6.6. Collapsing map

In this section, we define a mapping \( \text{coll} \) called the "collapsing map". Let \( T \) be a type theory based on TCS4. The map \( \text{coll} \) collapses \( T \) to the extended metalanguage, deleting all of the comonad-related structures of \( T \). For readability, we often write \( A^* \) for \( \text{coll}(A) \). From now on, we simply write "the metalanguage" to mean "the extended metalanguage".

First we define the collapsing map for the types of \( T \). Let \( \text{mltype} \) be a mapping which maps each basic type \( B \) of \( T \) to a type \( \text{mltype}(B) \) of the metalanguage. Then \( \text{coll}(A) \) is defined as the result of erasing all occurrences of \( L \) in \( A \) and replacing each occurrence of a basic type \( B \) by \( \text{mltype}(B) \). For the precise definition, see Table 8.

Next we define the collapsing map for the terms of \( T \). We assume that we are given a mapping \( \text{mlterm} \) which maps each constant symbol \( c \) of type \( A \) in \( T \) to a term \( \text{mlterm}(c) \) such that \( \vdash \text{mlterm}(c) : \text{coll}(A) \) is derivable in the metalanguage. Then the definition of \( \text{coll}(e) \) is given in Table 8.

Then we collapse judgments. The mapping \( \text{coll} \) collapses all occurrences of typecs and terms in judgments. The precise definition is given in Table 8.

Finally, we define the collapsing map for inference rules. Suppose that \( J_1, \ldots, J_n \rightarrow J \) is either a type formation rule or a type inference rule or an equality rule, where \( J_1, \ldots, J_n \) and \( J \) are judgments \( (n \geq 0) \). Then the collapsed image of this judgment is defined as
Collapsing types:

\[ A^- = \text{mtype}(A) \quad (A \text{ is a basic type}) \]
\[ 1^- = 1 \]
\[ (A \times B)^- = (A^- \times B^-) \]
\[ 0^- = 0 \]
\[ (A + B)^- = (A^- + B^-) \]
\[ (A \to B)^- = (A^- \to B^-) \]
\[ (LA)^- = A^- \]
\[ (MA)^- = MA^- \]

Collapsing terms:

\[ c^- = \text{mterm}(c) \quad (c \text{ is a constant}) \]
\[ x^- = x \quad (x \text{ is a variable}) \]
\[ *^- = * \]
\[ (e,d)^- = (e^-, d^-) \]
\[ (\text{inl}_{A,B}(e))^- = \text{inl}_{A^-,B^-}(e^-) \]
\[ (\text{inr}_{A,B}(d))^- = \text{inr}_{A^-,B^-}(d^-) \]
\[ (\text{case } e \text{ of } \text{inl}_{A,B}(x,y) \Rightarrow c \Rightarrow \text{inr}_{A,B}(y,d) \Rightarrow d^-) \]
\[ = (\text{case } e^- \text{ of } \text{inl}_{A^-,B^-}(x) \Rightarrow c^- \Rightarrow \text{inr}_{A^-,B^-}(y) \Rightarrow d^-) \]
\[ (\lambda x : A.e)^- = \lambda x : A^- . e^- \]
\[ (\text{box } f \text{ with } e_1, \ldots, e_n \text{ for } x_1, \ldots, x_n)^- \]
\[ = [e_1^-, \ldots, e_n^- / x_1, \ldots, x_n]f^- \]
\[ (\text{unbox } e)^- = e^- \]
\[ [e^-] = [e^-] \]
\[ (\text{let } x \Leftarrow e \text{ in } f \text{ with } e_1, \ldots, e_n \text{ for } x_1, \ldots, x_n)^- \]
\[ = (\text{let } x \Leftarrow e^- \text{ in } [e_1^-, \ldots, e_n^- / x_1, \ldots, x_n]f^-) \]
\[ (?^M_A(e))^- = ?^M_A(e^-) \]

Collapsing judgments:

\[ (\vdash A \text{ type})^- = (\vdash A^- \text{ type}) \]
\[ (\Gamma \vdash e : B)^- = (\Gamma^- \vdash e^- : B^-) \]
\[ (\Gamma \vdash e_1 =_A e_2)^- = (\Gamma^- \vdash (e_1^- =_A e_2^-)) \]

In the above, \( \Gamma^- \) means \( x_1 : A_1^-, \ldots, x_n : A_n^- \), if \( \Gamma \) is \( x_1 : A_1, \ldots, x_n : A_n \).

Lemma 6.9. A collapsed term \([e_1, \ldots, e_n / x_1, \ldots, x_n]e^-\) is identical to \([e_1^-, \ldots, e_n^- / x_1, \ldots, x_n]e^-,\) i.e. the collapsing map commutes with substitution.

Proof. Straightforward induction on the complexity of \( e \).

\[ J^- \]

Theorem 6.10. (1) If \( \vdash A \text{ type} \) is derived in \( T \), then \( (\vdash A \text{ type})^- \) is derived in the metalanguage.

(2) Each type inference rule of \( T \) is collapsed to a derived rule of the metalanguage.

(3) If \( \Gamma \vdash e : A \) is derived in \( T \), then \( (\Gamma \vdash e : A)^- \) is derived in the metalanguage.

(4) Each equality rule of \( T \) is collapsed to a derived rule of the metalanguage.

(5) If \( \Gamma \vdash e =_A e' \) is derived in \( T \), then \( (\Gamma \vdash e =_A e')^- \) is derived in the metalanguage.
Proof. (1) Clear.
(2) It is trivial for the cases other than the L.I, L.E and M.E rules.
- The L.I rule is collapsed to

\[
\Gamma^- \vdash e_i^- : A_i^- \quad (1 \leq i \leq n) \\
\Gamma^- \vdash x_1 : A_1^- , \ldots , x_n : A_n^- \vdash f^- : B^- \\
\Gamma^- \vdash \{e_1^- , \ldots , e_n^- / x_1 , \ldots , x_n\}f^- : B^-
\]

The lower judgment is derived from the upper judgments by substitution rule. Hence, this is a derived rule.
- The L.E rule is collapsed to

\[
\Gamma^- \vdash e^- : A^- \\
\Gamma^- \vdash e^- : A^-'
\]

which is a trivial rule.
- The M.E rule is collapsed to

\[
\Gamma^- \vdash e^- : MA^- \\
\Gamma^- \vdash e_i^- : A_i^- \quad (1 \leq i \leq n) \\
x_1 : A_1^- , \ldots , x_n : A_n^- , x : A^- \vdash f^- : MB^- \\
\Gamma^- \vdash (\text{let } x \leftarrow e^- \text{ in } [e_1^- , \ldots , e_n^- / x_1 , \ldots , x_n]f^- ) : MB^-'
\]

From the upper judgments, we can derive \( \Gamma^- , x : A^- \vdash [e_1^- , \ldots , e_n^- / x_1 , \ldots , x_n]f^- : MB^- \) . Thus, by the typing rule for let-expressions, we can derive the lower judgment.
(3) Immediate from (2).
(4) It is clear for the cases other than the rules for comonad and monad types.
- The cases of the rules box.\( \zeta \), unbox.\( \zeta \), [-].\( \zeta \) and let.\( \zeta \) are trivial.
- Consider the rules L.\( \beta \) and L.\( \eta \) . The lower judgment of L.\( \beta \) is collapsed to

\[
\Gamma^- \vdash [e_1^- , \ldots , e_n^- / x_1 , \ldots , x_n]f^- = B^- ((e_1 , \ldots , e_n / x_1 , \ldots , x_n)f)^- ,
\]

which is trivially derivable by the last lemma. The case of L.\( \eta \) is similar.
- Consider the rules box.ass.\( k \) and let.box.\( \text{comm} \).\( k \) . The lower judgment of the box.ass.\( k \) rule is collapsed to \( \Gamma^- \vdash h_1 = C^- h_2 \), where

\[
h_1 = [(\overline{d}_{1 \leq i < k})^- , (\overline{\varepsilon}_n)^- / \overline{x}_n]f^- , (\overline{d}_{k < i \leq m})^- / \overline{y}_m]^g^- ,
\]

\[
h_2 = [(\overline{d}_{1 \leq i < k})^- , (\overline{\varepsilon}_n)^- , (\overline{d}_{k < i \leq m})^- / \overline{y}_1 \leq i < k , \overline{x}_n , \overline{y}_k \leq i \leq m ]([\{\overline{x}_n / \overline{x}_n\}f^- / y_k]^g^- ) ,
\]

\[
(\overline{d}_{1 \leq i < k})^- = d_1^- , \ldots , d_k^- ,
\]

\[
(\overline{d}_{k < i \leq m})^- = d_{k+1}^- , \ldots , d_m^- .
\]

By the property of substitution, \( h_1 \) and \( h_2 \) are identical. Hence, the collapsed rule is trivially derivable. The case of the let.box.ass.\( k \) rule is similar.
- The let.ass rule is collapsed to

\[
\Gamma^- \vdash e^- : MA^- \\
\Gamma^- \vdash e_i^- : A_i^- \quad (1 \leq i \leq n)
\]
\[ \bar{x}_n: (\bar{A}_n)^-, x: A^- \vdash f^- : MB^- \quad \Gamma^- \vdash d_i^- : B_i^- \quad (1 \leq i \leq m) \]
\[ \bar{y}_m: (\bar{B}_m)^-, y: B^- \vdash g^- : MC^- \]
\[ \Gamma^- \vdash (\text{let } y \leftarrow ([\bar{c}_m^-]^{/\bar{x}_n} f^-) \text{ in } ([\bar{d}_m^-]^{/\bar{y}_m} g^-)) \]
\[ =_{MC^-} \quad (\text{let } x \leftarrow e^- \text{ in } (\text{let } y \leftarrow ([\bar{c}_m^-]^{/\bar{x}_n} f^-) \text{ in } ([\bar{d}_m^-]^{/\bar{y}_m} g^-)) \]

where \( \bar{x}_n: (\bar{A}_n)^- \) means \( x_1: A_1^-, \ldots, x_n: A_n^- \). From the upper judgments, we have\( \Gamma^-, x: A^- \vdash ([\bar{c}_n^-]^{/\bar{x}_n} f^-) \text{ and } \Gamma^-, y: B^- \vdash ([\bar{d}_m^-]^{/\bar{y}_m} g^-) \). Hence, we can derive the lower judgment by the ass rule of the metalanguage.

- The \( M.\beta \) rule is collapsed to
\[ \Gamma^- \vdash e^- : A^- \quad \Gamma^- \vdash e_i^- : A_i^- \quad (1 \leq i \leq n) \]
\[ \Gamma^- \vdash (\text{let } x \leftarrow e^- \text{ in } ([\bar{c}_n^-]^{/\bar{x}_n} f^-)) =_{MB^-} \quad [e^-, ([\bar{c}_n^-]^{/x} f^-)]. \]

From the upper judgments, we have \( \Gamma^-, x: A^- \vdash ([\bar{c}_n^-]^{/\bar{x}_n} f^-) \text{ and } \Gamma^-, y: B^- \vdash ([\bar{d}_m^-]^{/\bar{y}_m} g^-) \). Therefore, we can derive the lower judgment by the \( M.\beta \) rule of the metalanguage.

- The \( M.\eta \) rule is collapsed to
\[ \Gamma^- \vdash e^- : MA^- \quad \Gamma^- \vdash e_i^- : A_i^- \quad (1 \leq i \leq n) \]
\[ \Gamma^- \vdash (\text{let } x \leftarrow e^- \text{ in } [x]) =_{MA^-} \quad e^- \]

This rule is derived from the \( M.\eta \) rule of the metalanguage.

(5) Immediate from (4). \( \square \)

7. Modal logic proofs as programs

We shall discuss how to regard CS4 proofs as programs. We can answer this question as follows:

1. Interpret a deduction \( \Gamma \vdash A \) in CS4 (or a judgment \( \Gamma \vdash e : A \) in TCS4) in a suitable CS4 structure over some category \( \mathcal{C} \). Then we obtain an arrow \( f : [\Gamma] \to [A] \) in \( \mathcal{C} \). If we can regard arrows in \( \mathcal{C} \) as functional programs, then we can consider that we have derived a program from a proof.

2. We consider a metalanguage term as a program. Prove some judgment \( \Gamma \vdash e : A \) in TCS4, and collapse it to a metalanguage term.

However, these answers are too abstract. How can we write the specification of a program? How will the derived program work?

To give an answer to these questions, we define a realizability interpretation of CS4 and prove that we can extract a program from a proof using the interpretation.

In the following, we shall consider only extraction of side-effecting programs, because modal logics are suitable for reasoning on state-dependent propositions. Applications to other kind of programs are under investigation.

7.1. Generalized side-effect monad

To keep the argument as general as possible, we first introduce a slightly generalized notion of side-effect monad. In what follows, we write \( \bar{A} \) for \( \text{Hom}(1, A) \), the set of global elements of \( A \).
Definition 7.1. Let \( S \) be an arbitrary non-empty set. A generalized side-effect monad over \( S \) is a structure \( (\mathcal{C}, (M, \eta, \mu, t), R) \) satisfying the following conditions:

1. \( \mathcal{C} \) is a weakly co-cartesian ccc,
2. \( (M, \eta, \mu, t) \) is a strong monad over \( \mathcal{C} \),
3. \( R \) is a family of relations such that for each \( A \in \text{Obj}(C) \) and each \( e \in MA \), \( R_e \) is a binary relation of the form \( R_e \subseteq S \times (\tilde{A} \times S) \), and
4. \( R \) satisfies the following three conditions:

   (R1) \( R_{a, \eta_A} = \{(s,(a,s)) \mid s \in S\} \) for each \( A \in \text{Obj}(\mathcal{C}) \) and \( a \in \tilde{A} \),

   (R2) \( R_{a, f} = \{(s,(b,s''')) \mid \exists a'. \exists s'. (s,(a',s')) \in R_a \land (s',(b,s''')) \in R_{a';f} \} \) for any \( A,B \in \text{Obj}(\mathcal{C}) \) and \( f : A \rightarrow MB \),

   (R3) \( R_{(a,b);\mu_B} = \{(s,(c,s''')) \mid \exists b'. (s,(b',s''')) \in R_B \land c = \langle a,b' \rangle \} \) for any \( A,B \in \text{Obj}(\mathcal{C}) \), \( a \in \tilde{A} \) and \( b \in \tilde{MB} \).

Of course, the side-effect monads defined in Section 2 are naturally considered as generalized side-effect monads with obviously defined \( R \).

In what follows, we shall consider only generalized side effect monads of which monad functor preserves weak initials. An element of \( S \) is called a state. A global element of the form \( e : 1 \rightarrow MA \) is called a command, because it is thought of as a side-effecting computation. We adopt the following abbreviations:

\[
s \triangleright e \triangleright s' \iff \exists e'. (sR_e(e',s')), \quad s \triangleright s' \iff \exists A. \exists e \in M_A. (s \triangleright e \triangleright s').
\]

Lemma 7.2. (1) \( \triangleright \) is reflexive.

(2) \( \triangleright \) is transitive.

(3) \( R_{a, M(f)} = \{(s,((a'; f),s')) \mid (s,(a',s')) \in R_a \} \).

(4) \( R_{a, \mu_B} = \{(s,(b,s''')) \mid \exists a'. \exists s'. (s,(a',s')) \in R_a \land (s',(b,s''')) \in R_{a'} \} \).

Proof. (1) Let \( e = \eta_1 : 1 \rightarrow M1 \), then \( s \triangleright e \triangleright s \).

(2) Suppose that \( s \triangleright e \triangleright s' \) and \( s' \triangleright e' \triangleright s'' \) for some \( e \in MA \) and \( e' \in MB \). Let \( e'' = (e; (\vdash A; e')^*). \) Then \( s \triangleright e '' \triangleright s'' \).

(3), (4) Immediate from the conditions (R1) and (R2). \( \square \)

7.2. Definition of realizability

7.2.1. Informal discussion

First, we shall informally explain the ideas behind our realizability interpretation.

We think that \( \odot A \) means that: there exists a command \( e \) such that the execution of \( e \) terminates, and after every terminating execution of \( e \), \( A \) becomes true. Then a realizer of \( \odot A \) must give a command \( e \) and a realizer \( a \) which realizes \( A \) after the execution of \( e \). Since the truth of \( A \) depends on states and \( e \) may be non-deterministic, the realizer \( a \) must be chosen depending on how \( e \) has been executed. Accordingly, it is natural to think that \( a \) must be returned by \( e \) as the resulting value. That is, we
think that a realizer of $\Diamond A$ is a command $e \in MA$ which terminates and returns a realizer of $A$.

Next, we think that $\Box A$ means that: for all command $e$, after every terminating execution of $e.A$ becomes true. That is, $\Box A$ means that the truth of $A$ is never changed by any execution of any command. Then it is natural to think that a realizer of $\Box A$ at the state $s$ is such a that $a$ realizes $A$ at $s'$ for all $s'$ with $s \succ s'$. Since $\succ$ is reflexive, if $a$ realizes $\Box A$ at $s$, then $a$ also realizes $A$ at $s$. Therefore the type of a realizer of $\Box A$ must be the same as that of realizers of $A$.

Along these lines, we shall give the formal definition of realizability.

7.2.2. Formal definition
Suppose that a genealized side-effect monad $(\mathcal{C}, (M, \eta, \mu, t), R)$ is given. First we arbitrarily choose two mappings $p$ and $r$ so that $p(P)$ be an object of $\mathcal{C}$ and $r(s, P)$ be a subset of $p(P)$ for each propositional variable $P$ and state $s \in S$.

Type of realizers: Next we define $\langle \langle A \rangle \rangle_r$ for each proposition $A$ as follows:

$$
\langle \langle P \rangle \rangle_r = \rho(P) \quad (P \text{ is a propositional variable})
$$
$$
\langle \langle \top \rangle \rangle_r = 1 \quad \langle \langle \bot \rangle \rangle_r = 0
$$
$$
\langle \langle A \land B \rangle \rangle_r = \langle \langle A \rangle \rangle_r \times \langle \langle B \rangle \rangle_r \quad \langle \langle A \lor B \rangle \rangle_r = \langle \langle A \rangle \rangle_r + \langle \langle B \rangle \rangle_r
$$
$$
\langle \langle A \rightarrow B \rangle \rangle_r = \langle \langle A \rangle \rangle_r \Rightarrow \langle \langle B \rangle \rangle_r
$$
$$
\langle \langle \Box A \rangle \rangle_r = \langle \langle A \rangle \rangle_r \quad \langle \langle \Diamond A \rangle \rangle_r = M \langle \langle A \rangle \rangle_r.
$$

Intuitively, $\langle \langle A \rangle \rangle_r$ is the type of realizers of $A$. For $\Gamma = A_1, \ldots, A_n$, we define $\langle \langle \Gamma \rangle \rangle_r = \langle \langle A_1 \rangle \rangle_r \times \cdots \times \langle \langle A_n \rangle \rangle_r$.

Set of realizers: We define $\mathcal{R}_{\rho, r}(s, A)$ for each state $s$ and each proposition $A$ as follows:

$$
\mathcal{R}_{\rho, r}(s, P) = r(s, P) \quad (P \text{ is a propositional variable})
$$
$$
\mathcal{R}_{\rho, r}(s, \top) = \{id_1\} \quad (\text{singleton})
$$
$$
\mathcal{R}_{\rho, r}(s, \bot) = \emptyset \quad (\text{null set})
$$
$$
\mathcal{R}_{\rho, r}(s, A \land B) = \{\langle a, b \rangle \mid a \in \mathcal{R}_{\rho, r}(s, A) \land b \in \mathcal{R}_{\rho, r}(s, B)\}
$$
$$
\mathcal{R}_{\rho, r}(s, A \lor B) = \{a; \text{inl} \langle \langle A \rangle \rangle_r, \langle \langle B \rangle \rangle_r \mid a \in \mathcal{R}_{\rho, r}(s, A)\}
$$
$$
\cup\{b; \text{inr} \langle \langle A \rangle \rangle_r, \langle \langle B \rangle \rangle_r \mid b \in \mathcal{R}_{\rho, r}(s, B)\}
$$
$$
\mathcal{R}_{\rho, r}(s, A \rightarrow B) = \{e \in \langle \langle A \rightarrow B \rangle \rangle_r \mid \forall a \in \mathcal{R}_{\rho, r}(s, A). (e'a \in \mathcal{R}_{\rho, r}(s, B))\}
$$
$$
\mathcal{R}_{\rho, r}(s, \Box A) = \{e \in \langle \langle \Box A \rangle \rangle_r \mid \forall s' \in S. (s \succ s' \rightarrow e \in \mathcal{R}_{\rho, r}(s', A))\}
$$
$$
\mathcal{R}_{\rho, r}(s, \Diamond A) = \{e \in \langle \langle \Diamond A \rangle \rangle_r \mid (\exists s'. s \succ e' \land e' \forall s' (sR_e(e', s') \rightarrow e' \in \mathcal{R}_{\rho, r}(s', A))\}
$$

where $e'a$ is an abbreviation of $\langle e, a \rangle$; $eval$. Note that $\mathcal{R}_{\rho, r}(s, A)$ is defined as a subset of $\langle \langle A \rangle \rangle_r$. $\mathcal{R}_{\rho, r}(s, A)$ is the set of all realizers of $A$ at the state $s$. 

We write $er^s_A$ for $e \in \mathcal{R}_{p,r}(s,A)$ and read it as "$e$ realizes $A$ at the state $s$ (under the interpretation $(\rho,r)$)."

**Definition 7.3.** (1) We write $er^p_A$ to mean that $er^p_A$ holds for all states $s \in S$. It is read as "$e$ realizes $A$ (under the interpretation $(\rho,r)$)."

(2) A formula $A$ is realizable (under $(\rho,r)$) if and only if $er^p_A$ holds for some $e$.

(3) Let $f$ be an arrow, and let $A,\ldots,A_n$ and $A$ be CS4-formulas. Let $\Gamma = A_1,\ldots,A_n$. Then $f r^p_\Gamma (\Gamma \vdash A)$ holds, by definition, if and only if the following conditions hold:

(a) $f : \langle \Gamma \rangle_\rho \to \langle A \rangle_\rho$,

(b) for all $a_1 \in \langle \langle A_1 \rangle \rangle_\rho, \ldots,a_n \in \langle \langle A_n \rangle \rangle_\rho$, we have

$$\bigwedge_1 (a_1, r^p_{\Gamma} A_1) \land \cdots \land (a_n, r^p_{\Gamma} A_n) \Rightarrow ((a_1,\ldots,a_n); f) r^p_{\Gamma} A.$$

(4) We write $f r^p_\Gamma (\Gamma \vdash A)$ to mean that $f r^p_\Gamma (\Gamma \vdash A)$ holds for all states $s$.

(5) $\Gamma \vdash A$ is realizable (under $(\rho,r)$) if and only if we have $f r^p_\Gamma (\Gamma \vdash A)$ for some $f$.

Note that $f r^p_\Gamma (\Gamma \vdash A)$ is equivalent to $f r^p_{\Gamma} A$.

We write $\langle \langle A \rangle \rangle$, $\mathcal{R}(s,A)$, $r_s$ and $r$ for $\langle \langle A \rangle \rangle_\rho$, $\mathcal{R}_{p,r}(s,A), r^p_{\Gamma} A$ and $r^p_{\Gamma}$ respectively, when $\rho$ and $r$ are clear from the context.

### 7.3. Soundness of the realizability interpretation

We can prove the following theorem:

**Theorem 7.4.** Let $U$ be a theory based on CS4 and assume that all proper axioms of $U$ are realizable under $(\rho,r)$. Suppose that $\Gamma \vdash A$ is derivable in $U$. Then $\Gamma \vdash A$ is realizable under $(\rho,r)$.

**Proof.** We prove by induction on the derivation of $\Gamma \vdash A$. It is easy to see that all the axioms of intuitionistic logic are realizable and the intuitionistic rules preserve realizability. Accordingly, we shall consider the necessitation rule and the seven axioms on the modal operators.

- The necessitation rule. Assume that $e$ realizes $\Gamma \vdash A$. Then $er_s A$ for all $s$. Hence, if $s \triangleright s'$, we have $e r_s A$. Thus $e$ realizes $\Gamma \vdash A$.
- We show that $\Box-K$ is realizable. It is sufficient to show that $eval$ realizes $(\Box (A \to B),\Box A \vdash \Box B)$. Suppose $f r_s \Box (A \to B)$, $e r_s \Box (A)$ and $s \triangleright s'$. Then we have $f r_{s'} (A \to B)$ and $e r_{s'} A$. Hence $((f,e); eval) r_{s'} B$. Since $s'$ is arbitrary, we have $((f,e); eval) r_s \Box B$.
- $\Box-T$ and $\Box-4$. It is sufficient to show $id_{\langle A \rangle} r_s (\Box A \vdash A)$ and $id_{\langle A \rangle} r_s (\Box A \vdash \Box \Box A)$. These are obvious, because $\triangleright$ is reflexive and transitive.
- $\Diamond-K$. We show $t_{\langle \langle A \to B \rangle \rangle,\langle A \rangle} (eval) r_s (\Box A \vdash B), \Diamond A \vdash \Diamond B)$. Assume $f r_s \Box (A \to B)$ and $e r_s \Box A$. Then we have

$$\forall s' \in S. \forall a (s \triangleright s' \land (a r_{s'} A) \to f^* a r_{s'} B)$$
and

$$(\exists s'. s \succ s') \land \forall a \forall s'(s R_d(a, s') \to a r_s A).$$

Let $d = (f, e); r^I_{(\langle A \vdash B \rangle, \langle \langle A \rangle \rangle)}; M(eval)$. We must prove $d r_s \circ B$. We have

$$s R_d(b, s') \Rightarrow \exists a(s R_d(a, s') \land b = f^*a)$$

by Lemma 7.2. Therefore

$$(\exists s'. s \succ s') \land \forall b \forall s'(s R_d(b, s') \to b r_s B).$$

This shows that $d r_s \circ B$.

- $\circ_T$. It is easy to see that $\eta_{\langle A \rangle} r(A \vdash \circ A)$.

- $\circ_4$. We prove $\mu_{\langle A \rangle} r(\circ^2 A \vdash \circ A)$. Suppose $a r_s \circ^2 A$. Then

$$(\exists s'. s \succ s') \land \forall a' \forall s'(s R_d(a', s') \to$$

$$(\exists s''. s' \succ s'') \land \forall a'' \forall s''(s' R_d(a'', s'') \to a'' r_s A)).$$

holds. This implies

$$(\exists a' \exists s' \exists a'' \exists s''(s R_d(a', s') \land s' R_d(a'', s''))$$

$$\land \forall a' \forall s' \forall a'' \forall s''(s R_d(a', s') \land s' R_d(a'', s'') \to a'' r_s A).$$

Hence, by Lemma 7.2, we have

$$\left(\exists s'' \frac{a, \mu_{\langle A \rangle}}{s} \right) \land \forall a'' \forall s''(s R_d(a, s') \to a'' r_s A).$$

Therefore $a; \mu_{\langle A \rangle} r_s \circ A$.

- Assume that $e r_s \circ \bot$. Then $s R_d(e', s')$ and $e' \in R(s', \bot)$ for some $e'$ and $s'$. However, this is impossible, because $R(s', \bot)$ is empty. Therefore we never have $e r_s \circ \bot$. Hence $\eta^M_{\langle A \rangle} : M0 \to \langle \langle A \rangle \rangle$ realizes $\circ \vdash A$. $\square$

### 7.4. Collapsing map as program extraction

We show how to extract a metalanguage program from a proof.

**Definition 7.5.** Consider a metalanguage term $e$ with the typing judgment $A \vdash e : \tau$. Let $\Gamma \vdash A$ be a propositional sequent. Then we define

$$(A \vdash e : \tau)r^I_{\langle A \vdash \Gamma \vdash A \rangle} \iff [A \vdash e : \tau]^{I}_{\text{int}} r^I (\Gamma \vdash A)$$

where $[A \vdash e : \tau]^{I}_{\text{int}}$ is the interpretation of $A \vdash e : \tau$ under $I$ in the same strong monad as the one used to define $r^I$.

Note that if $(A \vdash e : \tau)r^I_{\langle \Gamma \vdash A \rangle}$ then $[A] = \langle \langle A \rangle \rangle_{\rho}$ and $[\tau] = \langle \langle \tau \rangle \rangle_{\rho}$ necessarily hold.
N.B. In the above definition, we used Moggi's semantics $[A \vdash e : \tau]^l_{ml}$ instead of our $\mathcal{L}^p$-strong monad semantics $[A \vdash e : \tau]_{ml, p}$. In this subsection, we are trying to consider a program as the collapsed image of a proof. Since collapsing map collapses comonad-related structures, it is natural not to use comonads here to interpret the program $e$. On the other hand, we will use an $\mathcal{L}^p$-strong monad in the next subsection to interpret proofs.

**Theorem 7.6.** Let $T$ be a type theory based on TCS4, $J$ be $(p, r)$, and $I$ be $(\Theta, \gamma)$. Assume that

1. for each basic type $A$ of $T$, we have $[A^\bot]^\Theta_{ml} = \langle \langle \text{Prop}(A) \rangle \rangle_p$, and
2. for any constant symbol $c$ of $T$, $\langle \langle c \rangle \rangle_p^l \text{Prop}(\langle \langle c \rangle \rangle_p)$

Then

$$(\Gamma \vdash e : A)^\gamma_p \text{Prop}(\Gamma \vdash e : A)$$

holds whenever $\Gamma \vdash e : A$ is derivable in $T$.

We prove this theorem in the next subsection using the categorical interpretation of TCS4, although it is not difficult to prove it directly.

**Program extraction:** This theorem shows that we can extract a program from a proof using the collapsing map. First we write a program specification of the form $\sqcap \diamond A$. It can be read as "It will always be true that there is a command which turns $A$ (constructively) true" or "Find a command which will always turn $A$ true". This formula corresponds to the type $LMA'$ by the mapping $\text{Type}$, where $A'$ means $\text{Type}(A)$. Then we prove $\vdash e : LMA'$ for some term $e$. When a proof of $\sqcap \diamond A$ is given, we can effectively find such $e$ using Theorem 6.2; otherwise, we may work in the type theory from the outset. If such $e$ is found, we can extract a program by collapsing $e$. By the last theorem, term $e^\gamma$ meets the given specification.

**7.5. Understanding realizability by $\mathcal{L}^p$-strong monad**

We have not used the notion of $\mathcal{L}^p$ strong monad in the definition of the realizability interpretation. However, it does not mean that $\mathcal{L}^p$-strong monads are irrelevant to the study of the realizability interpretation; on the contrary, our realizability interpretation is nothing but a special case of categorical semantics based on the notion of $\mathcal{L}^p$-strong monads.

Suppose that a generalized side-effect monad $(\mathcal{E}, (M, \eta, \mu, \tau), R)$ over $S$ is given and a realizability interpretation $r^\mathcal{E}_{\mathcal{E}}$ is defined. We define a category $\mathcal{E}$ as follows:

- An object $X$ is a function from $S$ to power set of $\tilde{A}$ for some $A \in \text{Obj}(\mathcal{E})$. The object $A$ is called the support of $X$, and written as $\text{supp} X$. Hence $X(s) \subseteq \text{supp} X$ for each state $s$ in $S$. We often write $\tilde{X}$ for $\text{supp} X$. 
An arrow from $X$ to $Y$ is a $\mathcal{C}$-arrow $f \in \mathcal{C}(\text{supp} X, \text{supp} Y)$ such that
$$\forall s \in S. \forall x \in X(s). ((x; f) \in Y(s)).$$

Composition is just composition in $\mathcal{C}$. Therefore, the identity $\tilde{id}_X$ on $X$ is the identity on $\text{supp} X$ in the sense of $\mathcal{C}$.

Then $\mathcal{C}$ has a natural CS4 structure $\mathcal{P} = (\mathcal{C}, (\tilde{\mathcal{M}}, \tilde{\mathcal{P}}, t^{\tilde{\mathcal{P}}}))$ defined as follows:

- **Structure of ccc:**
  - Terminal object $\tilde{1}$ is $\lambda s \in S. \{\text{id}_1\}$, terminal arrow $\tilde{1}_X$ is $!_X$.
  - Product $X \times Y = \lambda s \in S. \{(a, b) \in (\tilde{X} \times \tilde{Y}) \mid a \in X(s) \wedge b \in Y(s)\}$.
  - Projections are given by $\tilde{\pi}_{XY}^X = \pi_{\tilde{X}, \tilde{Y}}^X$ and $\tilde{\pi}_{XY}^Y = \pi_{\tilde{X}, \tilde{Y}}^Y$.
  - Product arrow $(f, g)$ is just $(f, g)$ in the sense of $\mathcal{C}$.
  - Exponential object $X \Rightarrow Y$ is
    $$\lambda s \in S. \{f \in \tilde{X} \Rightarrow \tilde{Y} \mid \forall x \in X(s). (f(x) \in Y(s))\}.$$

- $\tilde{A}f : Z \rightarrow (X \Rightarrow Y)$ (currying) is just $A f : \tilde{Z} \rightarrow (\tilde{X} \Rightarrow \tilde{Y})$.
- $\text{eval}_{X,Y} : (X \Rightarrow Y) \tilde{X} \times \tilde{Y} \rightarrow Y$ is $\text{eval}_{X,Y} : (X \Rightarrow Y) \times \tilde{X} \rightarrow \tilde{Y}$.

- **Weak co-cartesian structure:**
  - Weak initial $\tilde{0}$ is $\lambda s \in S. \emptyset$, where $\emptyset$ is considered as the null subset of $\tilde{0}$. Weak initial arrow $\tilde{0}_X$ is $?_X$.
  - Weak coproduct $X + Y$ is $\lambda s \in S. \{(x; \text{inl}_{X,Y} | x \in X(s)) \cup \{y; \text{inr}_{X,Y} | y \in Y(s)\}\}$
  - Injections are $\text{inl}_{X,Y} = \text{inl}_{X,Y}$ and $\text{inr}_{X,Y} = \text{inr}_{X,Y}$.
  - Weak coproduct arrow $[f, g] : X + Y \rightarrow Z$ is $[f, g] : \tilde{X} + \tilde{Y} \rightarrow \tilde{Z}$.

- **The cartesian comonad structure is defined by**
  - $\tilde{L}(X)$ is $\lambda s \in S. \{e \in \tilde{X} \mid \forall s' \in S. (s \rhd s' \rightarrow e \in X(s'))\}$. Note that $\text{supp} \tilde{L}(X)$ is the same as $\text{supp} X$.
  - Co-Kleisli lifting $f^L$ of $f$ is just $f$ itself.
  - $\tilde{m}_1 : \tilde{1} \rightarrow \tilde{L} \tilde{1}$ is $\text{id}_{\tilde{1}}$.
  - $\tilde{m}_{X,Y} : \tilde{L} \tilde{X} \tilde{\times} \tilde{L} \tilde{Y} \rightarrow \tilde{L}(\tilde{X} \times \tilde{Y})$ is the identity on $\tilde{X} \times \tilde{Y}$.

- **The monad structure is defined by**
  - $\tilde{M}(X)$ is $\lambda s \in S. \{e \in \tilde{M}(\tilde{X}) \mid (\exists s'. s \rhd s') \land \forall e' \forall s'(ssR_{\mathcal{C}}(e', s') \rightarrow e' \in X(s'))\}$. Note that $\text{supp} \tilde{M}(X)$ is $M(\text{supp} X)$.
  - $\tilde{\eta}_X$ is $\eta_{\tilde{X}}$.
  - Kleisli lifting $f^* \tilde{M}$ of $f : X \rightarrow \tilde{M} Y$ is just $f^* : M(\tilde{X}) \rightarrow M(\tilde{Y})$.

- **The $\tilde{P}$-tensorial strength for $\tilde{M}$ is given by $t_{X,Y}^{\tilde{P}} = t_{\tilde{X}, \tilde{Y}}$**.

- **The monad functor $\tilde{M}$ preserves weak initials. $?_{\tilde{X}}^M$ is $?_{\tilde{X}}^M$**.

It is not hard to verify that these definitions are well defined.

**Theorem 7.7.** Let $\tilde{\mathcal{C}}(P) = \lambda s \in S. \text{supp}_r(s, P)(= \lambda s \in S. r(s, P))$ and $\tilde{I}_U = (\tilde{\mathcal{P}}, \tilde{\omega})$.

1. For any $e : 1 \rightarrow (\langle A \rangle)_P$ in $\mathcal{C}$ and $s \in S$, we have
   $$er_{\tilde{r}}^s A \iff e \in [A]_{\text{CS4}}(s)$$
   (9)
\[ e^*_{\rho,r} A \iff e : \tilde{I} \rightarrow [A]^{\tilde{T}}_{\text{CS4}} \text{ in } \bar{\text{C}}. \] (10)

Hence, \( A \) is realizable under \((\rho, r)\) if and only if it is valid under \( \tilde{I}_U \).

2. For any \( f : \langle \langle \Gamma \rangle \rangle_\rho \rightarrow \langle \langle A \rangle \rangle_\rho \),

\[ f_{\rho,r}^* (\Gamma \vdash A) \iff \forall g \in \Gamma^{\tilde{T}}_{\text{CS4}}(s). (g, f \in [A]^{\tilde{T}}_{\text{CS4}}(s)), \]

\[ f_{\rho,r}^* (\Gamma \vdash A) \iff f : \Gamma^{\tilde{T}}_{\text{CS4}} \rightarrow [A]^{\tilde{T}}_{\text{CS4}} \text{ in } \bar{\text{C}}. \]

Hence, \( \Gamma \vdash A \) is realizable under \((\rho, r)\) if and only if it is valid under \( \tilde{I}_U \).

**Proof.** (1) (9) is easily proved by induction on the complexity of \( A \). (10) is clear from (9).

(2) Immediate from (1). \( \square \)

This result shows that our realizability interpretation is a special case of the categorical interpretation of propositional theories in CS4 structures.

### 7.5.1. Another proof of Theorem 7.4

One can prove Theorem 7.4 using the last lemma. Suppose \( \Gamma \vdash A \). Then, by Theorem 6.6, \( \Gamma \vdash A \) is valid under \( \tilde{I}_U \), and hence realizable.

### 7.5.2. Proof of Theorem 7.6

Let \( \tilde{\phi}(B) = \tilde{\omega}(\text{Prop}(B)) \) for each basic type \( B \), and let \( \tilde{\Theta} = (\tilde{\Phi}, \tilde{\omega}) \). Then:

**Lemma 7.8.** For each type \( A \), we have \( [A]^{\tilde{\Theta}}_{\text{CS4}} = [\text{Prop}(A)]^{\tilde{\Theta}}_{\text{CS4}} = \lambda s \in S. \mathcal{R}_{\rho,r}(s, \text{Prop}(A)) \) and \( \text{supp}[A]^{\tilde{\Theta}}_{\text{CS4}} = \langle \langle \text{Prop}(A) \rangle \rangle_\rho \).

**Proof.** By induction on the complexity of \( A \). Easy. \( \square \)

Suppose that the assumptions of Theorem 7.6 are satisfied. Let \( \tilde{\gamma}(c) = \langle \langle c : \text{typeof}(c) \rangle \rangle_{\text{ml}} \) and \( \tilde{I}_T = (\tilde{\Theta}, \tilde{\gamma}) \). Then:

**Lemma 7.9.** \( [A^{-}]^{\tilde{\Theta}}_{\text{ml}} = \langle \langle \text{Prop}(A) \rangle \rangle_\rho \) for any type \( A \). Hence, by Lemma 7.8, \( \text{supp}[A]^{\tilde{\Theta}}_{\text{CS4}} = [A^{-}]^{\tilde{\Theta}}_{\text{ml}} \).

**Proof.** Straightforward induction on the complexity of \( A \). \( \square \)

**Theorem 7.10.** (1) If \( \Gamma \vdash e : A \), then \( \Gamma \vdash e : A]^{\tilde{\Theta}}_{\text{CS4}} \phi^J \text{Prop}(\Gamma \vdash e : A) \).

(2) \( \Gamma \vdash e : A]^{\tilde{\Theta}}_{\text{CS4}} = \langle \langle \Gamma \vdash e : A \rangle \rangle_{\text{ml}} \).

**Proof.** (1) By Lemma 7.8, \( [\Gamma]^{\tilde{\Theta}}_{\text{CS4}} = [\text{Prop}(\Gamma)]^{\tilde{\Theta}}_{\text{CS4}} \). Therefore \( [\Gamma \vdash e : A]^{\tilde{\Theta}}_{\text{CS4}} : [\text{Prop}(\Gamma)]^{\tilde{\Theta}}_{\text{CS4}} \rightarrow [\text{Prop}(A)]^{\tilde{\Theta}}_{\text{CS4}} \). Hence, by Theorem 7.7, \( [\Gamma \vdash e : A]^{\tilde{\Theta}}_{\text{CS4}} \phi^J \text{Prop}(\Gamma \vdash e : A) \).
(2) We prove by induction on the derivation of \( \Gamma \vdash e : A \). For readability, we write \([A]^{\Theta}_{\text{CS4}}\) and \([A]^{\Theta}_{\text{mpl}}\) for \([A]_{\text{CS4}}^{\Theta}\) and \([A]_{\text{mpl}}^{\Theta}\), respectively. Similarly, we write \([\Gamma \vdash e : A]^{I}_{\text{CS4}}\) and \([\Gamma \vdash f : B]^{I}_{\text{mpl}}\) for \([\Gamma \vdash e : A]^{I}_{\text{CS4}}\) and \([\Gamma \vdash f : B]^{I}_{\text{mpl}}\), respectively. Consider the last rule applied. The cases of the rules for variables, products, coproducts and functions are easy.

- **The rule for constants:** For a constant \( c \) of type \( A \),

\[
[\Gamma \vdash c : A]^\gamma = [\Gamma]^\gamma \cdot \tilde{\gamma}(c) = \left[ (\Gamma \vdash c : A)^{-} \right] \quad \text{(by Lemma 7.9)}
\]

- **The L.I rule:** Let

\[
h_{i} = [\Gamma \vdash e_{i} : LA_{i}]^\gamma = [((\Gamma \vdash e_{i} : LA_{i})^{-}]
\]

\[g = [x_{1} : LA_{1}, \ldots, x_{n} : LA_{n} \vdash f : B]^\gamma = [(x_{1} : LA_{1}, \ldots, x_{n} : LA_{n} \vdash f : B)^{-}].\]

Then

\[
[\Gamma \vdash \text{box } f \text{ with } e_{1}, \ldots, e_{n} \text{ for } x_{1}, \ldots, x_{n} : LB]^\gamma = \\
\left[ (\Gamma \vdash (\text{box } f \text{ with } e_{1}, \ldots, e_{n} \text{ for } x_{1}, \ldots, x_{n}) : LB)^{-} \right]
\]

- **The L.E rule:**

\[
[\Gamma \vdash \text{unbox } e : A]^\gamma = [\Gamma \vdash e : LA]^\gamma \cdot \tilde{\varepsilon}_{[A]} \quad \text{(by I.H. and the definition of } \tilde{\varepsilon})
\]

\[
[\Gamma \vdash \text{unbox } e : A)^{-} = [((\Gamma \vdash e : LA)^{-}])^{-1} \cdot \eta_{[A]}^{-1} \quad \text{(by I.H. and the definition of } \tilde{\eta})
\]

- **The M.I rule:**

\[
[\Gamma \vdash [e] : MA]^\gamma = [\Gamma \vdash e : A]^\gamma \cdot \tilde{\eta}_{[A]} = [((\Gamma \vdash e : A)^{-}])^{-1} \cdot \eta_{[A]}^{-1} \quad \text{(by I.H. and the definition of } \tilde{\eta})
\]

- **The M.E rule:** Suppose

\[
g_{1} = [\Gamma \vdash e : MA]^\gamma = [((\Gamma \vdash e : MA)^{-}]
\]

\[
h_{i} = [\Gamma \vdash e_{i} : LA_{i}]^\gamma = [((\Gamma \vdash e_{i} : LA_{i})^{-}]
\]

\[
g_{2} = [\tilde{x}_{n} : LA_{n}, x \vdash f : MB]^\gamma = [((\tilde{x}_{n} : LA_{n}, x \vdash f : MB)^{-}].
\]
Then

\[
[\Gamma \vdash (\text{let } x \leftarrow e \text{ in } f \text{ with } \tilde{e}_n \text{ for } \tilde{x}_n) : MB]\]

\[
= \langle (\langle h_1, \ldots, h_n \rangle; m_{A_1[x\ldots x\{A_n\}]}, g_1)\rangle^*; q_{[A_1[x\ldots x\{A_n\}]}; ([m_{A_1[x\ldots x\{A_n\}]}])^{-1} \times id_{[A_1]}; g_2 \rangle^*
\]

\[
= \langle (h_1, \ldots, h_n), g_1; t_{[A_1[x\ldots x\{A_n\}]}}; k_{[A_1[x\ldots x\{A_n\}]i} g_2^* \rangle
\]

\[
= \langle id_{[A_1[x\ldots x\{A_n\}]i}, g_1; t_{[A_1[x\ldots x\{A_n\}]i}M((h_1, \ldots, h_n) \times id_{[A_1]}); g_2 \rangle^*
\]

\[
= \langle id_{[A_1[x\ldots x\{A_n\}]i}, g_1; t_{[A_1[x\ldots x\{A_n\}]i}((h_1, \ldots, h_n) \times id_{[A_1]}); g_2 \rangle^*
\]

\[
= [\Gamma \vdash \text{let } x \leftarrow e^{-} \text{ in } [e_1^{-}, \ldots, e_n^{-}/x_1, \ldots, x_n]f^{-} : MB^{-}]
\]

\[
= [\Gamma \vdash (\text{let } x \leftarrow e \text{ in } f \text{ with } \tilde{e}_n \text{ for } \tilde{x}_n) : MB^{-}].
\]

**The MO.E rule:**

\[
[\Gamma \vdash \gamma^M_A(e) : A] = [\Gamma \vdash e : M0] = [\Gamma \vdash e : M0]^{*}; [A]^{*}
\]

\[
= [\langle (\Gamma \vdash e : M0)^{-}; \gamma^M_A \rangle_{[A]^{-}}
\]

\[
= [\Gamma \vdash \gamma^M_A(e) : A].
\]

This completes the proof. □

Now we can prove Theorem 7.6 easily. Assume that \( \Gamma \vdash e : A \) in \( T \). Then, by the above theorem, \( [(\Gamma \vdash e : A)^{-}]_{\text{inf}}r^J \text{Prop}(\Gamma \vdash e : A) \) holds, and hence we have

\( (\Gamma \vdash e : A)^{-}r^J \text{Prop}(\Gamma \vdash e : A) \).

8. A toy example of program extraction

We shall present a toy example of program extraction.

Let us consider the following problem: There is a room with an electric light with a toggle switch. A robot with a sensor eye is in the room. By reading the output of the sensor, he can know whether the room is light or dark. The task for the robot is to turn the light on.

This problem is modeled as follows: Let \( S \) be the two-point set \{light,dark\}. Consider the deterministic side-effect monad over \textbf{Set} with \( S \) as the set of states. We identify an element of \( A \) (in set-theoretical sense) and the corresponding global element in \( \text{Hom}(1,A) \). We write \( * \) for the unique element of \( 1 \). Define \( \text{Bool} \in \text{Obj}(	extbf{Set}) \) by

\( \text{Bool} = 1 + 1 = \{\text{inl}_{1,1}(\ast), \text{inr}_{1,1}(\ast)\} \).
We write \textit{true} for $\text{inl}_{1,1}(\ast)$ and \textit{false} for $\text{inr}_{1,1}(\ast)$. We define $\text{sensor} \in M \text{Bool}$ and $\text{toggle} \in M 1$ by

\begin{align*}
\text{sensor} &= \{(\text{light},(\text{true},\text{light})),(\text{dark},(\text{false},\text{dark}))\}, \\
\text{toggle} &= \{(\text{light},(\ast,\text{dark})),(\text{dark},(\ast,\text{light}))\}.
\end{align*}

Consider the two propositional variables \text{Light} and \text{Dark}. Let

\begin{align*}
\rho(\text{Light}) = \rho(\text{Dark}) = 1 & \in \text{Obj}(\text{Set}), \\
r(\text{light}, \text{Light}) = 1, & \quad r(\text{dark}, \text{Light}) = \emptyset, \\
r(\text{light}, \text{Dark}) = \emptyset, & \quad r(\text{dark}, \text{Dark}) = 1.
\end{align*}

Consider the metalanguage with two constants $\text{sensor} : M \text{Bool}$ and $\text{toggle} : M 1$, where $\text{Bool} = 1 + 1$. We define

\begin{align*}
\textit{if} \ a \ \textit{then} \ b \ \textit{else} \ c \ \textit{case} \ \text{a} \ \textit{of} \ \textit{inl}_{1,1}(x) \ \Rightarrow b \ | \ \textit{inr}_{1,1}(y) \ \Rightarrow c \ (x \notin \text{FV}(b), y \notin \text{FV}(c))
\end{align*}

\text{skip} = [\ast].

Define \(\gamma(\text{sensor}) = \text{sensor}\) and \(\gamma(\text{toggle}) = \text{toggle}\). Then

\begin{align*}
[\vdash \text{sensor} : M \text{Bool}]_{\text{ml} r^m r^\square} & \circ (\text{Light} \lor \text{Dark}), \\
[\vdash \lambda x : \text{toggle} : 1 \rightarrow M 1]_{\text{ml} r^m r^\square} & (\text{Dark} \rightarrow \circ \text{Light}), \\
[\vdash \lambda x : \text{toggle} : 1 \rightarrow M 1]_{\text{ml} r^m r^\square} & (\text{Light} \rightarrow \circ \text{Dark}).
\end{align*}

Let

\begin{align*}
U = CS4 + \square \circ (\text{Light} \lor \text{Dark}) + \square (\text{Dark} \rightarrow \circ \text{Light}) + \square (\text{Light} \rightarrow \circ \text{Dark}).
\end{align*}

Then, by Theorem 7.4, all provable formulas of \(U\) are realizable. For simplicity, we identify a propositional variable \(P\) and the corresponding basic type $\text{Type}(P)$. $\text{FF}(U)$ is $\text{TCS}4(C)$ with $C = \{\text{AxSensor}, \text{AxToggle1}, \text{AxToggle2}\}$, where

\begin{align*}
\text{AxSensor} : L(M(\text{Light} + \text{Dark})), \\
\text{AxToggle1} : L(\text{Dark} \rightarrow M \text{Light}), \\
\text{AxToggle2} : L(\text{Light} \rightarrow M \text{Dark}).
\end{align*}

Here we have written

\begin{align*}
\text{AxSensor} \quad \text{for} \quad \text{AxConst}(\square \circ (\text{Light} \lor \text{Dark})), \\
\text{AxToggle1} \quad \text{for} \quad \text{AxConst}(\square (\text{Dark} \rightarrow \circ \text{Light})), \\
\text{AxToggle2} \quad \text{for} \quad \text{AxConst}(\square (\text{Light} \rightarrow \circ \text{Dark})).
\end{align*}
We define
\[
mltype(\text{Light}) = mltype(\text{Dark}) = 1, \\
mlterm(\text{AxSensor}) = \text{sensor}, \\
mlterm(\text{AxToggle1}) = mlterm(\text{AxToggle2}) = \lambda x : 1.\text{toggle}.
\]

Then all the assumptions of Theorem 7.6 are satisfied.

The specification of the program for the robot's task is written as \( \Box \Diamond \text{Light} \). It can be read as "It is always the case that there is a command which turns the light on", or "Find a command which will always turn the light on". This specification formula corresponds to the type \( LMLight \) by the mapping \( \text{Type} \). We must find a term \( e \) and prove \( \vdash e : LMLight \) in the type theory \( \mathcal{T}(U) \). Let
\[
e = \text{box}(\text{let } z \leftarrow \text{unbox } \text{AxSensor} \text{ in case } z \text{ of } \text{inl}_{1,1}(u) \\
\Rightarrow [u]|\text{inr}_{1,1}(v) \Rightarrow (\text{unbox } w)v \text{ with } \text{AxToggle1} \text{ for } w).
\]

Then the reader can easily verify that \( \vdash e : LMLight \) is derivable in \( \mathcal{T}(U) \). Let us calculate the collapsing of \( e \):
\[
e^- = \text{let } z \leftarrow \text{sensor} \text{ in case } z \text{ of } \text{inl}_{1,1}(u) \Rightarrow [u]|\text{inr}_{1,1}(v) \Rightarrow (\lambda x : 1.\text{toggle})v \\
= \text{let } z \leftarrow \text{sensor} \text{ in case } z \text{ of } \text{inl}_{1,1}(u) \Rightarrow [\ast]|\text{inr}_{1,1}(v) \Rightarrow \text{toggle} \\
\text{(by } 1.\eta \text{ and } \rightarrow \beta) \\
= \text{let } z \leftarrow \text{sensor} \text{ in if } z \text{ then skip else toggle}.
\]

This is the extracted program that meets the specification.

9. Conclusion and future work

We can consider two kinds of type systems: one can be viewed as a natural logic, and the other cannot be. Moggi's type system is of the latter kind. It cannot be a natural modal logic. If we want to make it natural, we must make some modifications and add comonad types to it. In this way, we obtain the type system TCS4 that encodes the constructive version of S4 modal logic.

It is usually said that a constructive proof is a program. However, exactly speaking, we should say that a constructive proof consists of a program and its correctness proof. The correctness part is not necessary at runtime. The realizability interpretation cuts off the correctness part (to some extent) and extracts the program part. When we describe the specification of a program or prove a certain theorem in modal logic, we need the \( \Box \)-modality in general, because \( \Box \) is needed for correctness proofs. However, the comonad types that model \( \Box \)-modality are not necessarily needed in the type system for the extracted programs, because "the correctness part" is no longer needed.
We meet a similar situation when considering dependent types. Constructive type systems designed for proof development usually have dependent products \( \Pi x.A(x) \) and dependent sums \( \sum x.A(x) \) for encoding of quantification. However, we can “collapse” these types to ordinary function types and ordinary product types when we are extracting programs from proofs. Dependent types are not necessarily needed in type systems for runtime programs. We think that a comonad type \( \mathcal{L}A \) is collapsed to \( A \) for the same reason. The collapsing map collapses the comonad-related structures of a term and extracts a program.

Our toy example is Section 8 is the simplest one for illustration of program extraction. For more complex problems such as in-place sorting of an array, we will need a predicate modal logic. The author thinks that extending our result to the predicate logic version of CS4 is not difficult. However, it will need much effort to develop a method to extract efficient programs. It will be our future work.

Our logic has only two modal operators \( \Box \) and \( \Diamond \). Therefore, it is restricted in expressiveness. To increase the expressive power of our logic, the author is now studying a type theory with evaluation modalities. For details of evaluation modalities, see Pitt’s [9] paper.

10. Related works

Bierman and de Paiva [1] studied a constructive S4 logic without \( \Diamond \) modality. They proposed Hilbert style, sequent calculus, and natural deduction formulations of their logic and gave a term-assignment system. Our type theory TCS4 is a natural extension of their term-assignment system. They also gave a categorical semantics of their logic using monoidal comonads. They did not use monads, because their logic did not have \( \Diamond \)-modality. Our semantics for CS4 is an extension of their semantics except that we used cartesian comonads while they used monoidal ones. Every cartesian comonad is of course monoidal, but the converse is not true. The reason why they did not need cartesian comonads is that they did not consider \( \eta \)-equalities for box-expressions.

Wijesekera [13] studied a constructive version of \( K \) modal logic, which is a subsystem of CS4. He gave a Kripke semantics for his logic and proved its soundness and completeness. He also proved that his system enjoys cut elimination.

Brookes and Geva [2] introduced the notion of a computational comonad and used it to model intensional aspects of computations. Our \( \mathcal{L} \)-strong monad semantics of the metalanguage can be seen as a unification of their semantics and Moggi’s one. However, computational comonads cannot model any meaningful \( \Box \)-modality, because every computational comonad must have a natural transformation \( \gamma : \text{Id}_c \rightarrow L \) and hence validates \( A \leftrightarrow \Box A \).

In [8], a function from \( A \) to \( B \) possibly invoking continuations is interpreted as a proof of a linear logic formula of the form \( !A - \Diamond !B \). Recall that our new semantics of the metalanguage interprets a function from \( A \) to \( MB \) as a morphism from \( L[A] \) to \( ML[B] \). The reader will find the similarity between \( !A - \Diamond !B \) and \( LA \rightarrow MLB \). Nishizaki’s work
and ours are probably related, although the exact relationship between them has not been studied yet.

Acknowledgements

The author wishes to thank the participants of SLACS’93–’94 meetings and NSL’94 workshop, the attendants of Seta Seminar and the anonymous referee for their helpful comments, suggestions and criticisms. In particular, discussions and e-mail exchanges with Mr. Masahito Hasegawa were fruitful. He also pointed out Refs. [1,2] to the author. Thanks are also due to Mr. Kenji Taguchi for providing information on the work of Wijesekera [13].

This research was partially supported by a grant from the Ministry of Education of Japan.

Appendix. Proof of Proposition 6.8

We abbreviate $x : A \vdash e_1 =_B e_2$ as $e_1 = e_2$ when the omitted parts are not significant. First we prove the following lemma:

Lemma A.1. We have the following equalities:

1. $\Box f$ with $\bar{\epsilon}_n, g, g, \bar{d}_m$ for $\bar{x}_n, z_1, z_2, \bar{y}_m = \Box [z, z/z_1, z_2]f$ with $\bar{\epsilon}_n, g, \bar{d}_m$ for $\bar{x}_n, z, \bar{y}_m$ (box-contraction).

2. $\Box f$ with $\bar{\epsilon}_n, g_1, g_2, \bar{d}_m$ for $\bar{x}_n, z_1, z_2, \bar{y}_m = \Box f$ with $\bar{\epsilon}_n, g_2, g_1, \bar{d}_m$ for $\bar{x}_n, z_2, z_1, \bar{y}_m$ (box-exchange).

3. $\Box f$ with $\bar{\epsilon}_n$ for $\bar{x}_n = \Box f$ with $\bar{\epsilon}_n, d$ for $\bar{x}_n, y$ (if $y$ is not free in $f$), (box-weakening).

4. let $x \Leftarrow e$ in $f$ with $\bar{\epsilon}_n, d, d, \bar{d}_m$ for $\bar{x}_n, z_1, z_2, \bar{y}_m = \text{let } x \Leftarrow e \text{ in } [z, z/z_1, z_2]f$ with $\bar{\epsilon}_n, d, \bar{d}_m$ for $\bar{x}_n, z, \bar{y}_m$ (let-contraction).

5. let $x \Leftarrow e$ in $f$ with $\bar{\epsilon}_n, g_1, g_2, \bar{d}_m$ for $\bar{x}_n, z_1, z_2, \bar{y}_m = \text{let } x \Leftarrow e \text{ in } f$ with $\bar{\epsilon}_n, g_2, g_1, \bar{d}_m$ for $\bar{x}_n, z_2, z_1, \bar{y}_m$ (let-exchange).

6. let $x \Leftarrow e$ in $f$ with $\bar{\epsilon}_n$ for $\bar{x}_n = \text{let } x \Leftarrow e \text{ in } f$ with $\bar{\epsilon}_n, d$ for $\bar{x}_n, y$ (if $y$ is not free in $f$), (let-weakening).

Proof. We prove (1),(2),(3) and (5). The other equalities are left to the reader.

(1)

$\Box f$ with $\bar{\epsilon}_n, g, g, \bar{d}_m$ for $\bar{x}_n, z_1, z_2, \bar{y}_m$

$= \Box (\text{unbox } (\Box f \text{ with } \bar{x}_n, z, \bar{y}_m \text{ for } \bar{x}_n, z_1, z_2, \bar{y}_m))$ with $\bar{\epsilon}_n, g, \bar{d}_m$

for $\bar{x}_n, z, \bar{y}_m$ (by L.η)

$= \Box [z, z/z_1, z_2]f$ with $\bar{\epsilon}_n, g, \bar{d}_m$ for $\bar{x}_n, z, \bar{y}_m$ (by L.β)
(2) 
\[ \text{box } f \text{ with } \bar{e}_n, g_1, g_2, \bar{d}_m \text{ for } \bar{x}_n, z_1, z_2, \bar{y}_m \]
\[ = \text{box (unbox (box } f \text{ with } \bar{x}_n, z_2, z_1, \bar{y}_m \text{ for } \bar{x}_n, z_2, z_1, \bar{y}_m)) \]
\[ \text{with } \bar{e}_n, g_1, g_2, \bar{d}_m \text{ for } \bar{x}_n, z_1, z_2, \bar{y}_m \text{ (by } L. \beta) \]
\[ = \text{box } f \text{ with } \bar{e}_n, g_2, g_1, \bar{d}_m \text{ for } \bar{x}_n, z_2, z_1, \bar{y}_m \text{ (by } L. \eta) \]

(3) 
\[ \text{box } f \text{ with } \bar{e}_n \text{ for } \bar{x}_n \]
\[ = \text{box (unbox (box } f \text{ with } \bar{x}_n \text{ for } \bar{x}_n)) \text{ with } \bar{e}_n, d \text{ for } \bar{x}_n, y \text{ (by } L. \eta) \]
\[ = \text{box } f \text{ with } \bar{e}_n, d \text{ for } \bar{x}_n, y \text{ (by } L. \beta) \]

(5) We let 
\[ h_i = \text{box } \pi_i(\text{unbox } w) \text{ with } z \text{ for } w \text{ (for } i = 1, 2), \]
\[ h = \text{box } \langle \text{unbox } u_1, \text{unbox } u_2 \rangle \text{ with } z_1, z_2 \text{ for } u_1, u_2, \]
\[ h' = \text{box } \langle \text{unbox } u_1, \text{unbox } u_2 \rangle \text{ with } z_2, z_1 \text{ for } u_2, u_1, \]
\[ g = [h_1, h_2/z_1, z_2] f. \]

Then we have 
\[ h = h', [h/z] h_i = z_i (i = 1, 2), \] and 
\[ f = [h/z] g = [h'/z] g. \] Hence,
\[ \text{let } x \Leftarrow e \text{ in } f \text{ with } \bar{e}_n, g_1, g_2, \bar{d}_m \text{ for } \bar{x}_n, z_1, z_2, \bar{y}_m \]
\[ = \text{let } x \Leftarrow e \text{ in } [h/z] g \text{ with } \bar{e}_n, g_1, g_2, \bar{d}_m \text{ for } \bar{x}_n, z_1, z_2, \bar{y}_m \]
\[ = \text{let } x \Leftarrow e \text{ in } g \text{ with } \bar{e}_n, [g_1, g_2/z_1, z_2] h, \bar{d}_m \text{ for } \bar{x}_n, z, \bar{y}_m \]
\[ \text{ (by } \text{let.box.comm.}(n + 1)) \]
\[ = \text{let } x \Leftarrow e \text{ in } g \text{ with } \bar{e}_n, [g_1, g_2/z_1, z_2] h', \bar{d}_m \text{ for } \bar{x}_n, z, \bar{y}_m \]
\[ = \text{let } x \Leftarrow e \text{ in } [h'/z] g \text{ with } \bar{e}_n, g_2, g_1, \bar{d}_m \text{ for } \bar{x}_n, z_1, z_2, \bar{y}_m \]
\[ \text{ (by } \text{let.box.comm.}(n + 1)) \]
\[ = \text{let } x \Leftarrow e \text{ in } f \text{ with } \bar{e}_n, g_2, g_1, \bar{d}_m \text{ for } \bar{x}_n, z_2, z_1, \bar{y}_m \]

Using the above lemma, we prove that \( \mathcal{S} \) is a CS4 structure. We shall prove only that (i) \( m_1 \) is an isomorphism; (ii) \( m_{A,B} \) is an isomorphism, and (iii) \( (id_{LA} \times \mu_B); t_{A,B}^L = t_{A,MB}^L; M(t_{A,B}^L); \mu_{LA \times B}. \) Verification of the other conditions are left to the reader.
(i) It is sufficient to show $x : L1 \vdash \text{box} * = L1 x$:

$$\text{box} * = \text{box} * \text{ with } x \text{ for } z \quad \text{(by box.weakening)}$$

$$= \text{box} (\text{unbox } z) \text{ with } x \text{ for } z \quad \text{(by }\lambda.\eta)$$

$$= [x/z]z \quad \text{(by }L.\eta)$$

$$= x.$$

(ii) First we prove $m_{A_1,A_2}; L(n_i) = n_i$.

$$\text{box } \pi_i \text{ (unbox } x) \text{ with }$$

\begin{align*}
\text{box} (\text{unbox } y_1, \text{unbox } y_2) \text{ with } & \pi_1(x), \pi_2(x) \text{ for } y_1, y_2 \text{ for } x \\
= \text{box} (\{\text{box} (\text{unbox } y_1, \text{unbox } y_2) \text{ with } z_1, z_2 \text{ for } y_1, y_2/x\} \pi_i(\text{unbox } x)) \\
& \text{ with } \pi_1(x), \pi_2(x) \text{ for } z_1, z_2 \quad \text{(by box.ass.1)} \\
= \text{box} (\pi_i(\text{unbox} (\text{box} (\text{unbox } y_1, \text{unbox } y_2) \text{ with } z_1, z_2 \text{ for } y_1, y_2))) \\
& \text{ with } \pi_1(x), \pi_2(x) \text{ for } z_1, z_2 \\
= \text{box} (\pi_i(\text{unbox } z_1, \text{unbox } z_2)) \text{ with } \pi_1(x), \pi_2(x) \text{ for } z_1, z_2 \quad \text{(by }L.\beta) \\
= \text{box} (\text{unbox } z_i) \text{ with } \pi_1(x), \pi_2(x) \text{ for } z_1, z_2 \quad \text{(by }\times.\beta) \\
= \pi_i(x) \quad \text{(by }L.\eta). \\
\end{align*}

Hence $m_{A_1,A_2}; (L(\pi_1), L(\pi_2)) = id_{L(A_1) \times L(A_2)}$.

Next we verify $(L(\pi_1), L(\pi_2)); m_{A_1,A_2} = id_{L(A_1 \times A_2)}$.

$$\text{box} (\text{unbox } y_1, \text{unbox } y_2) \text{ with } \pi_1((g_1(x), g_2(x))), \pi_2((g_1(x), g_2(x))) \text{ for } y_1, y_2$$

$$= \text{box} (\text{unbox } y_1, \text{unbox } y_2) \text{ with } g_1(x), g_2(x) \text{ for } y_1, y_2$$

$$= \text{box} ([g_1(z_1), g_2(z_2)/y_1, y_2]\{\text{unbox } y_1, \text{unbox } y_2\}) \text{ with } x, x \text{ for } z_1, z_2$$

$$\text{(by box.ass.1 and box.ass.2)}$$

$$= \text{box} ([\text{unbox } g_1(z_1), \text{unbox } g_2(z_2)]) \text{ with } x, x \text{ for } z_1, z_2$$

$$= \text{box} ([\text{unbox } g_1(z), \text{unbox } g_2(z)]) \text{ with } x \text{ for } z \quad \text{(by box.contraction)}$$

$$= \text{box} (\pi_1(\text{unbox } z), \pi_2(\text{unbox } z)) \text{ with } x \text{ for } z \quad \text{(by }L.\beta)$$

$$= \text{box} (\text{unbox } z) \text{ with } x \text{ for } z \quad \text{(by }\times.\eta)$$

$$= x \quad \text{(by }L.\eta)$$
(iii) We must prove \((x : LA \times M^2B \vdash e_1 = M(\lambda x y. e_2))\), where

\[
e_1 = (\text{let } w \leftarrow (\text{let } y \leftarrow \pi_2(x) \text{ in } y) \text{ in } [(u, w)] \text{ with } \pi_1(x) \text{ for } u),
\]

\[
e_2 = (\text{let } r \leftarrow (\text{let } z \leftarrow (\text{let } y \leftarrow \pi_2(x) \text{ in } [(v, y)] \text{ with } \pi_1(x) \text{ for } v)
\]

\[
in [\text{let } w \leftarrow \pi_2(z) \text{ in } [(u, w)] \text{ with } \pi_1(z) \text{ for } u]) \text{ in } r).
\]

By the \text{let} ass rule,

\[
e_1 = \text{let } y \leftarrow \pi_2(x) \text{ in } (\text{let } w \leftarrow y \text{ in } [(u, w)] \text{ with } v \text{ for } u) \text{ with } \pi_1(x) \text{ for } v.
\]

Further,

\[
e_2 = \text{let } r \leftarrow (\text{let } y \leftarrow \pi_2(x) \text{ in } (\text{let } z \leftarrow [(v, y)] \text{ in } [\text{let } w \leftarrow \pi_2(z) \text{ in } [(u, w)]
\]

\[
\text{with } \pi_1(z) \text{ for } u]) \text{ with } \pi_1(x) \text{ for } v) \text{ in } r \text{ (by let} ass)
\]

\[
= \text{let } r \leftarrow (\text{let } y \leftarrow \pi_2(x) \text{ in } [\text{let } w \leftarrow y \text{ in } [(u, w)] \text{ with } v \text{ for } u]
\]

\[
\text{with } \pi_1(x) \text{ for } v) \text{ in } r \text{ (by } M.\beta \text{ and } \times.\beta)
\]

\[
= \text{let } y \leftarrow \pi_2(x) \text{ in } (\text{let } r \leftarrow [\text{let } w \leftarrow y \text{ in } [(u, w)] \text{ with } v \text{ for } u] \text{ in } r)
\]

\[
\text{with } \pi_1(x) \text{ for } v \text{ (by let} ass)
\]

\[
= \text{let } y \leftarrow \pi_2(x) \text{ in } (\text{let } w \leftarrow y \text{ in } [(u, w)] \text{ with } v \text{ for } u) \text{ with } \pi_1(x)
\]

\[
\text{for } v \text{ (by } M.\beta).
\]

Therefore, \(e_1 = e_2\). □

References


